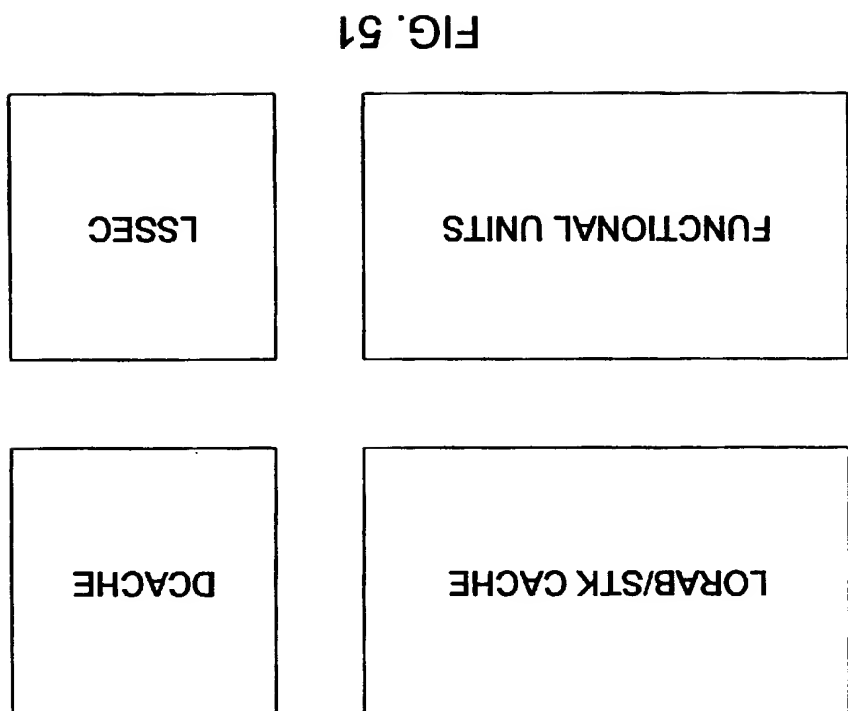


	L #	Hits	Search Text	DBs
1	L1	114	(plural plurality multiple multiplicity several two second) adj2 (pc pa ip ((instruction program) adj2 (counter address))) near30 (multithread\$3 thread\$3 context)	USPAT; US-PGPUB
2	L2	2859	(plural plurality multiple multiplicity several two second) adj2 (pc pa ip ((instruction program) adj2 (counter address))) and (multithread\$3 thread\$3 context)	USPAT; US-PGPUB
3	L3	683	(select\$3 interleav\$3 execut\$3) near10 ((cycl\$6 robin) near20 (multithread\$3 thread\$3 context))	USPAT; US-PGPUB
4	L4	137	(fine adj2 grain\$3) near20 (multithread\$3 thread\$3 context)	USPAT; US-PGPUB
5	L5	86	2 and (3 4)	USPAT; US-PGPUB
6	L6	63	1 not 5	USPAT; US-PGPUB
7	L7	7	(plural plurality multiple multiplicity several two second) adj2 (pc pa ip ((instruction program) adj2 (counter address))) near30 (multithread\$3 thread\$3 context)	EPO; JPO; DERWENT; IBM_TDB
8	L8	22	(plural plurality multiple multiplicity several two second) adj2 (pc pa ip ((instruction program) adj2 (counter address))) and (multithread\$3 thread\$3 context)	EPO; JPO; DERWENT; IBM_TDB
9	L9	61	(select\$3 interleav\$3 execut\$3) near10 ((cycl\$6 robin) near20 (multithread\$3 thread\$3 context))	EPO; JPO; DERWENT; IBM_TDB
10	L10	28	(fine adj2 grain\$3) near20 (multithread\$3 thread\$3 context)	EPO; JPO; DERWENT; IBM_TDB



	Docum ent ID	U	Title	Current OR
1	US 20040 07377 8 A1	<input type="checkbox"/>	Parallel processor architecture	712/220
2	US 20040 05488 0 A1	<input checked="" type="checkbox"/>	Microengine for parallel processor architecture	712/245
3	US 20030 18956 5 A1	<input checked="" type="checkbox"/>	Single semiconductor graphics platform system and method with skinning, swizzling and masking capabilities	345/418
4	US 20030 16366 9 A1	<input checked="" type="checkbox"/>	Configuration of multi-cluster processor from single wide thread to two half-width threads	712/24
5	US 20030 14988 8 A1	<input checked="" type="checkbox"/>	Integrated network intrusion detection	713/200
6	US 20030 14515 9 A1	<input checked="" type="checkbox"/>	SRAM controller for parallel processor architecture	711/104
7	US 20030 11224 6 A1	<input checked="" type="checkbox"/>	Blending system and method in an integrated computer graphics pipeline	345/519
8	US 20030 11224 5 A1	<input checked="" type="checkbox"/>	Single semiconductor graphics platform	345/506
9	US 20030 11036 6 A1	<input checked="" type="checkbox"/>	Run-ahead program execution with value prediction	712/225
10	US 20030 10590 1 A1	<input checked="" type="checkbox"/>	PARALLEL MULTI-THREADED PROCESSING	710/240
11	US 20030 10562 0 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for interface constructs in a programming language capable of programming hardware architectures	703/22
12	US 20030 10305 4 A1	<input checked="" type="checkbox"/>	Integrated graphics processing unit with antialiasing	345/506
13	US 20030 10305 0 A1	<input checked="" type="checkbox"/>	Masking system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
14	US 20030 09754 8 A1	<input checked="" type="checkbox"/>	Context execution in pipelined computer processor	712/228
15	US 20030 07417 7 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for a simulator plug-in for co-simulation purposes	703/22
16	US 20030 04667 1 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for signal constructs in a programming language capable of programming hardware architectures	717/141
17	US 20030 04666 8 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for distributing IP cores	717/131

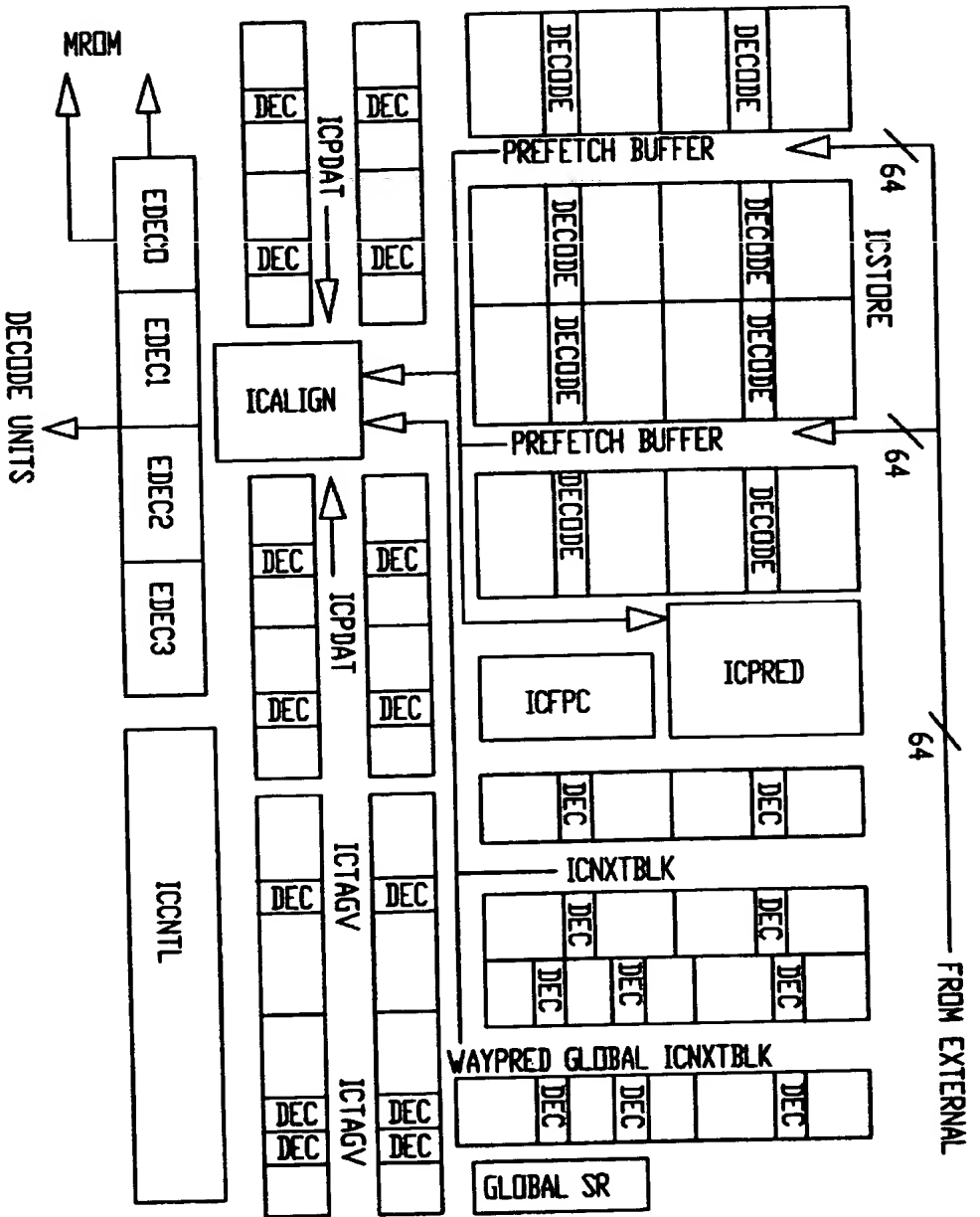


FIG. 27

	Docum ent ID	U	Title	Current OR
18	US 20030 03880 8 A1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/506
19	US 20030 03732 1 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for extensions in a programming lanauage capable of programming hardware architectures	717/149
20	US 20030 03497 5 A1	<input checked="" type="checkbox"/>	Lighting system and method for a graphics processor	345/426
21	US 20030 03359 4 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for parameterized expression libraries	717/141
22	US 20030 03358 8 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for using a library map to create and maintain IP cores effectively	717/107
23	US 20030 02886 4 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for successive compilations using incomplete parameters	717/141
24	US 20030 02072 0 A1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/506
25	US 20030 00526 2 A1	<input checked="" type="checkbox"/>	Mechanism for providing high instruction fetch bandwidth in a multi-threaded processor	712/207
26	US 20020 19917 3 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for a debugger capable of operating across multiple threads and lock domains	717/129
27	US 20020 19625 9 A1	<input checked="" type="checkbox"/>	Single semiconductor graphics platform with blending and fog capabilities	345/506
28	US 20020 18074 0 A1	<input checked="" type="checkbox"/>	Clipping system and method for a single graphics semiconductor platform	345/506
29	US 20020 12922 7 A1	<input checked="" type="checkbox"/>	Processor having priority changing function according to threads	712/228
30	US 20020 10551 9 A1	<input checked="" type="checkbox"/>	Clipping system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
31	US 20020 09191 5 A1	<input checked="" type="checkbox"/>	Load prediction and thread identification in a multithreaded microprocessor	712/225
32	US 20020 07812 1 A1	<input checked="" type="checkbox"/>	Real-time scheduler	718/102
33	US 20020 06600 5 A1	<input checked="" type="checkbox"/>	Data processor with an improved data dependence detector	712/218
34	US 20020 05603 7 A1	<input checked="" type="checkbox"/>	Method and apparatus for providing large register address space while maximizing cycletime performance for a multi-threaded register file set	712/215

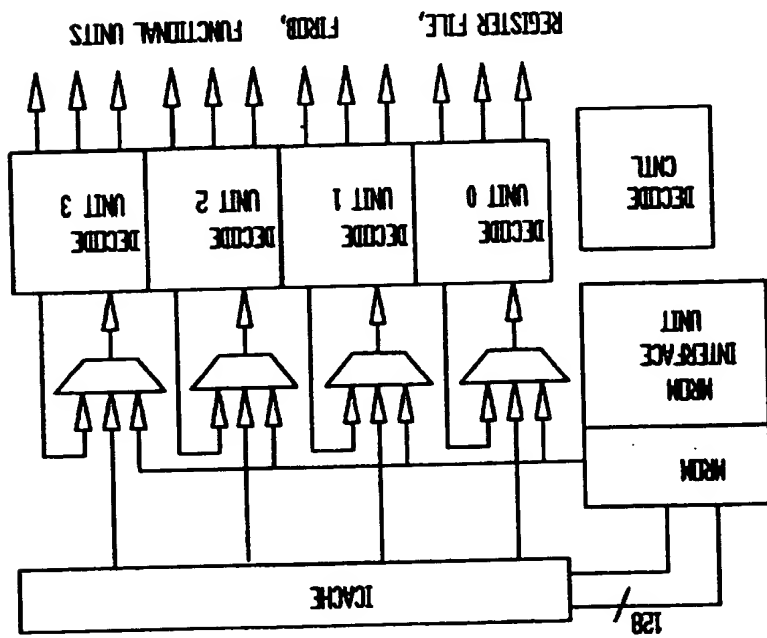


FIG. 28

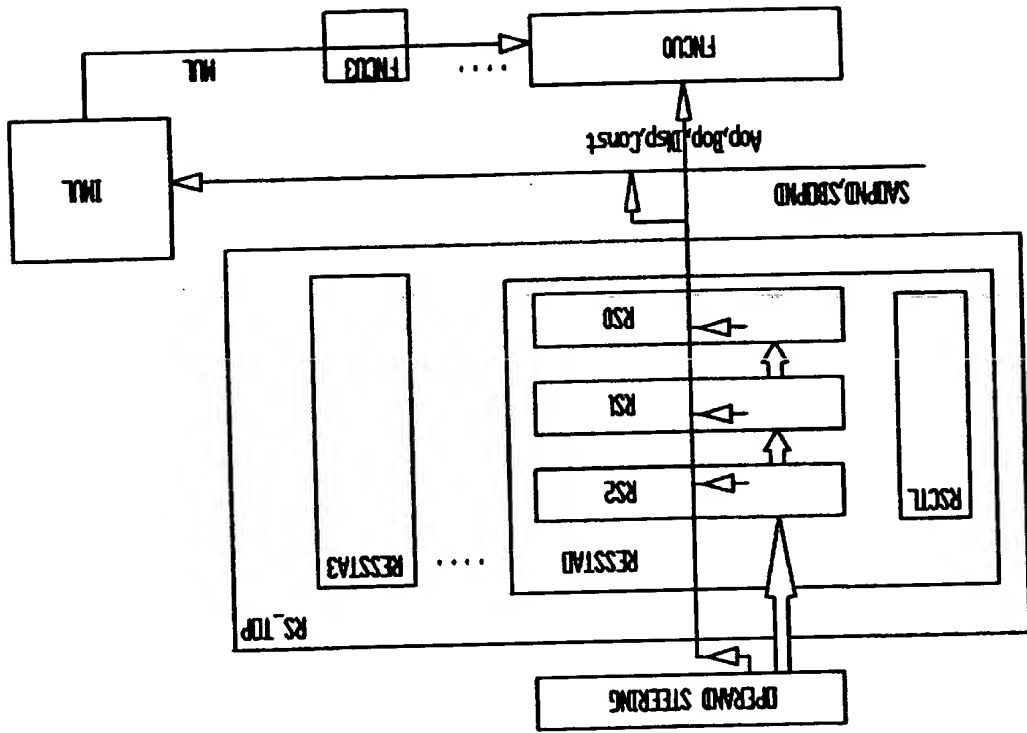


FIG. 29

	Docum ent ID	U	Title	Current OR
35	US 20020 04784 6 A1	<input checked="" type="checkbox"/>	System, method and computer program product for performing a scissor operation in a graphics processing framework embodied on a single semiconductor platform	345/522
36	US 20020 04632 5 A1	<input checked="" type="checkbox"/>	Buffer memory management in a system having multiple execution entities	711/122
37	US 20020 02755 3 A1	<input checked="" type="checkbox"/>	Diffuse-coloring system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
38	US 20020 01386 1 A1	<input checked="" type="checkbox"/>	Method and apparatus for low overhead multithreaded communication in a parallel processing environment	719/313
39	US 20020 01079 3 A1	<input checked="" type="checkbox"/>	METHOD AND APPARATUS FOR PERFORMING FRAME PROCESSING FOR A NETWORK	709/240
40	US 20010 05645 6 A1	<input checked="" type="checkbox"/>	PRIORITY BASED SIMULTANEOUS MULTI-THREADING	718/103
41	US 20010 05205 3 A1	<input checked="" type="checkbox"/>	Stream processing unit for a multi-streaming processor	711/138
42	US 20010 04977 0 A1	<input checked="" type="checkbox"/>	BUFFER MEMORY MANAGEMENT IN A SYSTEM HAVING MULTIPLE EXECUTION ENTITIES	711/129
43	US 20010 04746 8 A1	<input checked="" type="checkbox"/>	Branch and return on blocked load or store	712/228
44	US 20010 03744 5 A1	<input checked="" type="checkbox"/>	Cycle count replication in a simultaneous and redundantly threaded processor	712/216
45	US 20010 01762 6 A1	<input checked="" type="checkbox"/>	Graphics processing unit with transform module capable of handling scalars and vectors	345/501
46	US 20010 00520 9 A1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a transform module in a graphics processor	345/506
47	US 66913 01 B2	<input checked="" type="checkbox"/>	System, method and article of manufacture for signal constructs in a programming language capable of programming hardware architectures	717/114
48	US 66683 17 B1	<input checked="" type="checkbox"/>	Microengine for parallel processor architecture	712/245
49	US 66584 47 B2	<input checked="" type="checkbox"/>	Priority based simultaneous multi-threading	718/103
50	US 66503 31 B2	<input checked="" type="checkbox"/>	System, method and computer program product for performing a scissor operation in a graphics processing framework embodied on a single semiconductor platform	345/522
51	US 66503 30 B2	<input checked="" type="checkbox"/>	Graphics system and method for processing multiple independent execution threads	345/506
52	US 66503 25 B1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for boustrophedonic rasterization	345/426
53	US 66256 54 B1	<input checked="" type="checkbox"/>	Thread signaling in multi-threaded network processor	709/230

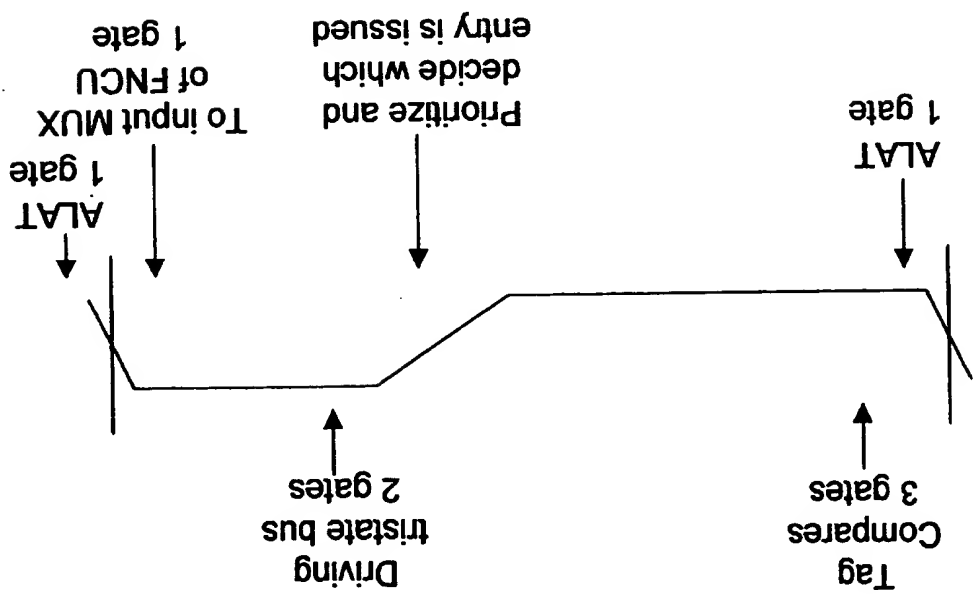


FIG. 30

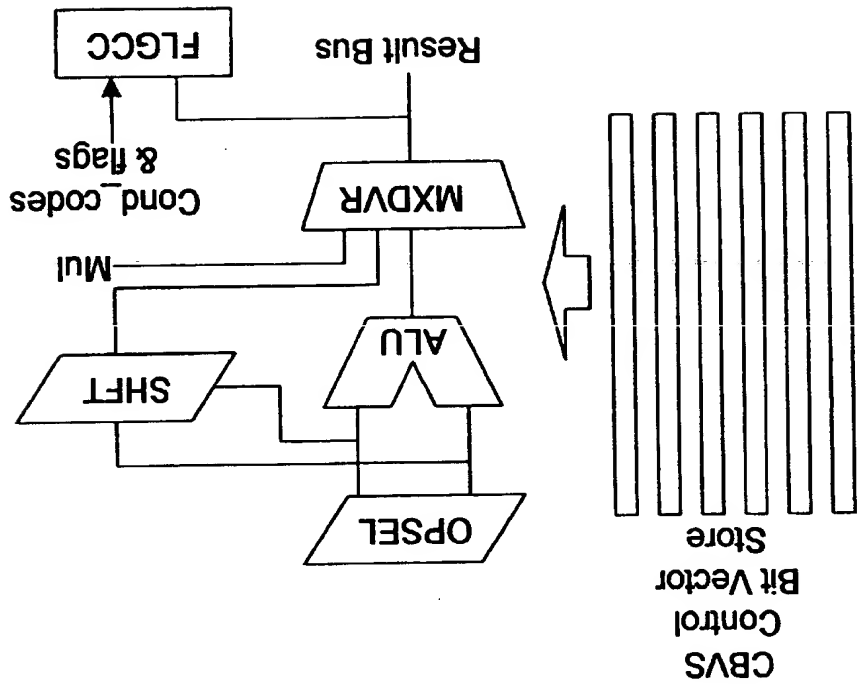


FIG. 31

	Docum ent ID	U	Title	Current OR
54	US 66067 04 B1	<input checked="" type="checkbox"/>	Parallel multithreaded processor with plural microengines executing multiple threads each microengine having loadable microcode	712/248
55	US 65973 56 B1	<input checked="" type="checkbox"/>	Integrated tessellator in a graphics processing unit	345/423
56	US 65879 06 B2	<input checked="" type="checkbox"/>	Parallel multi-threaded processing	710/240
57	US 65781 37 B2	<input checked="" type="checkbox"/>	Branch and return on blocked load or store	712/228
58	US 65773 09 B2	<input checked="" type="checkbox"/>	System and method for a graphics processing framework embodied utilizing a single semiconductor platform	345/426
59	US 65739 00 B1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/537
60	US 65642 67 B1	<input checked="" type="checkbox"/>	Network adapter with large frame transfer emulation	709/250
61	US 65325 09 B1	<input checked="" type="checkbox"/>	Arbitrating command requests in a parallel multi-threaded processing system	710/240
62	US 65156 71 B1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a vertex attribute buffer in a graphics processor	345/506
63	US 65045 42 B1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for area rasterization using sense points	345/441
64	US 64704 43 B1	<input checked="" type="checkbox"/>	Pipelined multi-thread processor selecting thread instruction in inter-stage buffer based on count information	712/205
65	US 64704 22 B2	<input checked="" type="checkbox"/>	Buffer memory management in a system having multiple execution entities	711/129
66	US 64627 37 B2	<input checked="" type="checkbox"/>	Clipping system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
67	US 64525 95 B1	<input checked="" type="checkbox"/>	Integrated graphics processing unit with antialiasing	345/426
68	US 64271 96 B1	<input checked="" type="checkbox"/>	SRAM controller for parallel processor architecture including address and command queue and arbiter	711/158
69	US 64178 51 B1	<input checked="" type="checkbox"/>	Method and apparatus for lighting module in a graphics processor	345/426
70	US 63779 98 B2	<input checked="" type="checkbox"/>	Method and apparatus for performing frame processing for a network	709/236
71	US 63743 67 B1	<input checked="" type="checkbox"/>	Apparatus and method for monitoring a computer system to guide optimization	714/37
72	US 63634 75 B1	<input checked="" type="checkbox"/>	Apparatus and method for program level parallelism in a VLIW processor	712/206
73	US 63534 39 B1	<input checked="" type="checkbox"/>	System, method and computer program product for a blending operation in a transform module of a computer graphics pipeline	345/561
74	US 63493 63 B1	<input checked="" type="checkbox"/>	Multi-section cache with different attributes for each section	711/129
75	US 63428 88 B1	<input checked="" type="checkbox"/>	Graphics processing unit with an integrated fog and blending operation	345/426
76	US 63306 61 B1	<input checked="" type="checkbox"/>	Reducing inherited logical to physical register mapping information between tasks in multithread system using register group identifier	712/228

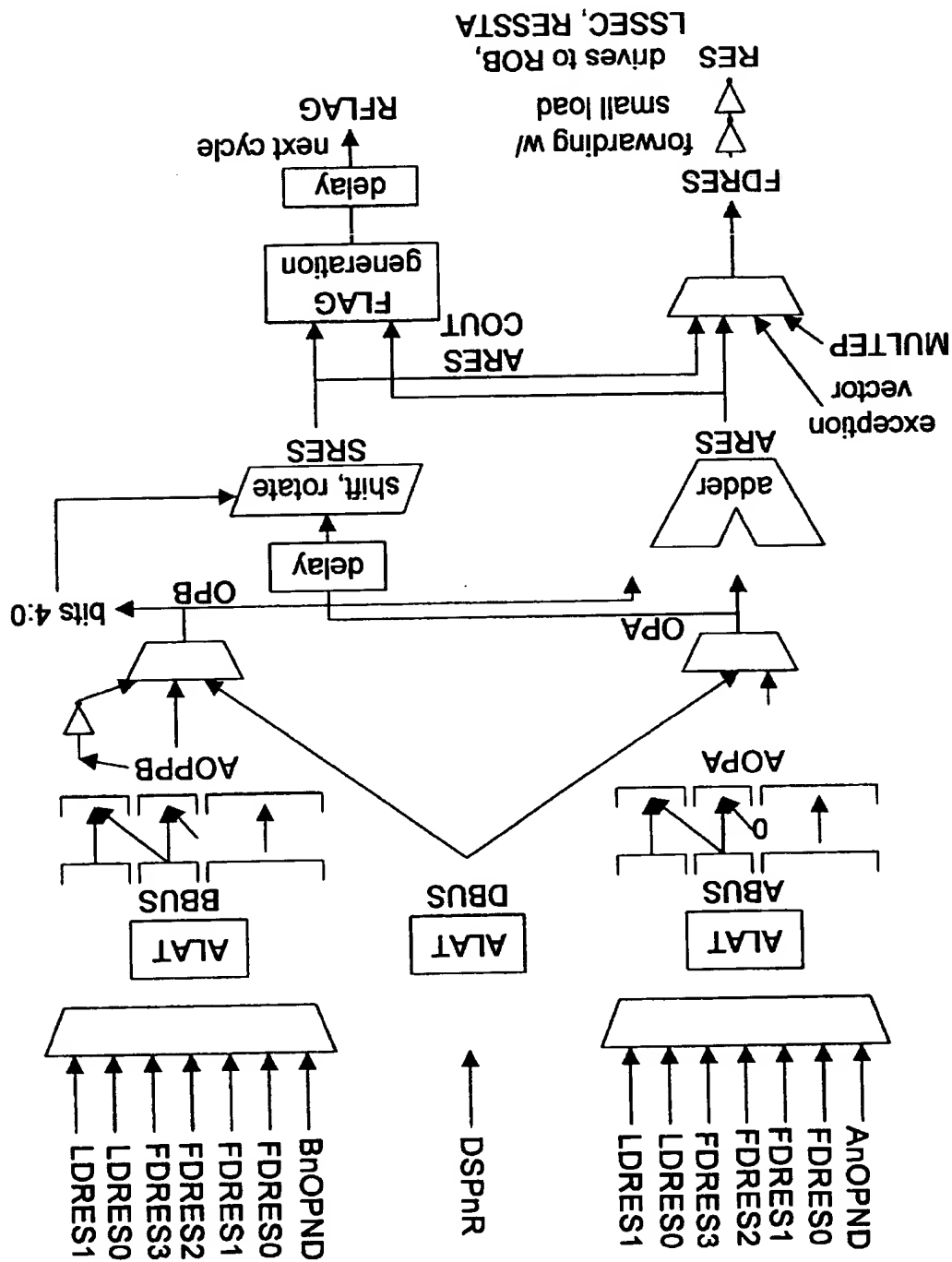


FIG. 32

	Docum ent ID	U	Title	Current OR
77	US 62956 00 B1	<input checked="" type="checkbox"/>	Thread switch on blocked load or store using instruction thread field	712/228
78	US 62894 46 B1	<input checked="" type="checkbox"/>	Exception handling utilizing call instruction with context information	712/244
79	US 62162 20 B1	<input checked="" type="checkbox"/>	Multithreaded data processing method with long latency subinstructions	712/219
80	US 61984 88 B1	<input checked="" type="checkbox"/>	Transform, lighting and rasterization system embodied on a single semiconductor platform	345/426
81	US 61700 51 B1	<input checked="" type="checkbox"/>	Apparatus and method for program level parallelism in a VLIW processor	712/225
82	US 60731 59 A	<input checked="" type="checkbox"/>	Thread properties attribute vector based thread selection in multithreading processor	718/103
83	US 59499 94 A	<input checked="" type="checkbox"/>	Dedicated context-cycling computer with timed context	712/228
84	US 59336 27 A	<input checked="" type="checkbox"/>	Thread switch on blocked load or store using instruction thread field	712/228
85	US 58549 22 A	<input checked="" type="checkbox"/>	Micro-sequencer apparatus and method of combination state machine and instruction memory	712/245
86	US 55749 22 A	<input checked="" type="checkbox"/>	Processor with sequences of processor instructions for locked memory updates	712/220

	Docum ent ID	U	Title	Current OR
1	US 20040 06224 5 A1	<input type="checkbox"/>	TCP/IP offload device	370/392
2	US 20040 05500 3 A1	<input checked="" type="checkbox"/>	Uniprocessor operating system design facilitating fast context swtiching	718/108
3	US 20040 04250 7 A1	<input checked="" type="checkbox"/>	Method and apparatus for fast change of internet protocol headers compression mechanism	370/521
4	US 20040 03470 8 A1	<input checked="" type="checkbox"/>	Method and apparatus for fast internet protocol headers compression initialization	709/227
5	US 20040 03087 3 A1	<input checked="" type="checkbox"/>	Single chip multiprocessing microprocessor having synchronization register file	712/245
6	US 20030 22968 3 A1	<input checked="" type="checkbox"/>	Information providing system and method and storage medium	709/219
7	US 20030 15430 7 A1	<input checked="" type="checkbox"/>	Method and apparatus for aggregate network address routes	709/245
8	US 20030 14515 5 A1	<input checked="" type="checkbox"/>	Data transfer mechanism	711/104
9	US 20030 09357 1 A1	<input checked="" type="checkbox"/>	Information providing system and method and storage medium	709/248
10	US 20030 07454 2 A1	<input checked="" type="checkbox"/>	Multiprocessor system and program optimizing method	712/20
11	US 20030 06021 0 A1	<input checked="" type="checkbox"/>	System and method for providing real-time and non-real-time services over a communications system	455/452 .1
12	US 20030 02623 0 A1	<input checked="" type="checkbox"/>	Proxy duplicate address detection for dynamic address allocation	370/338
13	US 20030 01447 2 A1	<input checked="" type="checkbox"/>	Thread ending method and device and parallel processor system	718/107
14	US 20030 00974 3 A1	<input checked="" type="checkbox"/>	METHOD AND APPARATUS FOR PRE-PROCESSING AND PACKAGING CLASS FILES	717/117
15	US 20020 19161 2 A1	<input checked="" type="checkbox"/>	Method and apparatus for automatically determining an appropriate transmission method in a network	370/392
16	US 20020 18604 6 A1	<input checked="" type="checkbox"/>	Circuit architecture for reduced-synchrony on-chip interconnect	326/47
17	US 20020 18144 8 A1	<input checked="" type="checkbox"/>	Prevention of spoofing in telecommunications systems	370/352

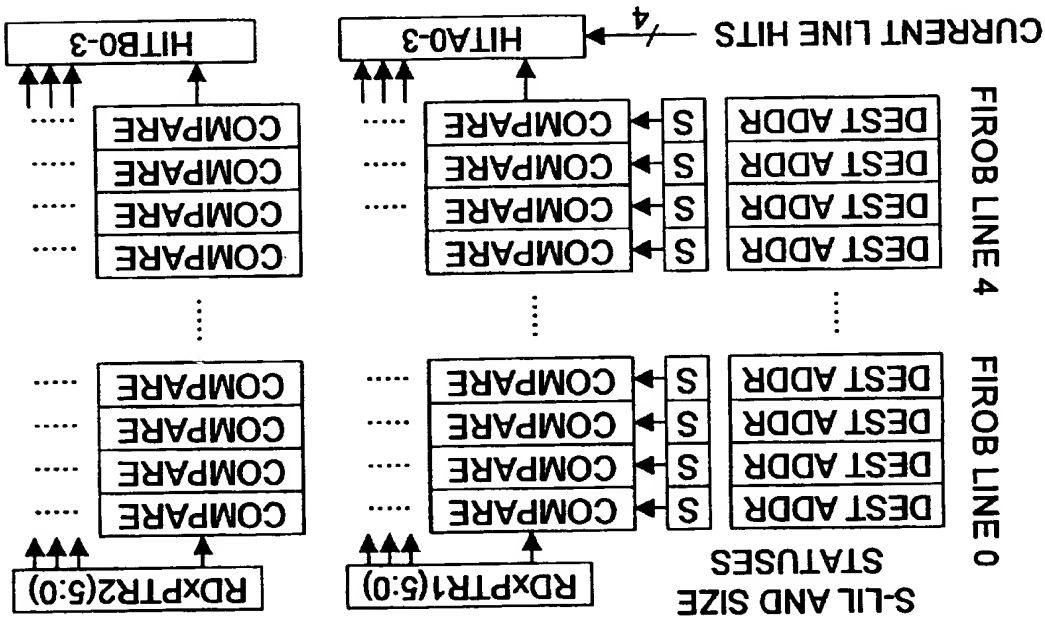


FIG. 35

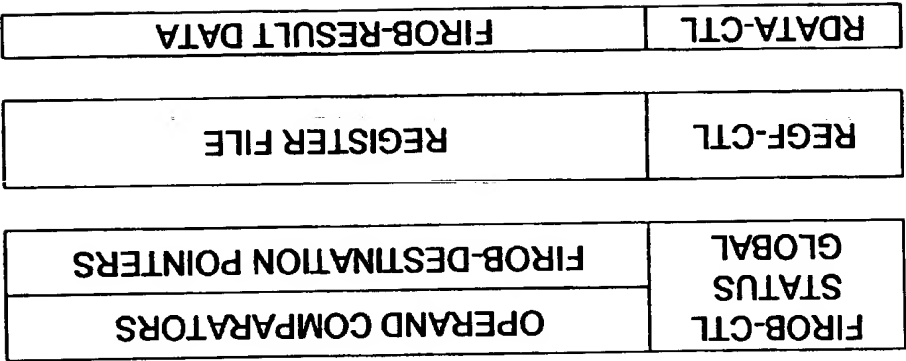


FIG. 36

	Docum ent ID	U	Title	Current OR
18	US 20020 16980 4 A1	<input checked="" type="checkbox"/>	System and method for storage space optimized memorization and generation of web pages	715/513
19	US 20020 09984 4 A1	<input checked="" type="checkbox"/>	Load balancing and dynamic control of multiple data streams in a network	709/232
20	US 20020 05596 4 A1	<input checked="" type="checkbox"/>	Software controlled pre-execution in a multithreaded processor	718/107
21	US 20020 04543 7 A1	<input checked="" type="checkbox"/>	Tracing a location of a mobile device	455/411
22	US 20020 02320 2 A1	<input checked="" type="checkbox"/>	Load value queue input replication in a simultaneous and redundantly threaded processor	712/225
23	US 20020 01071 0 A1	<input checked="" type="checkbox"/>	Method for characterizing a complex system	715/500
24	US 20010 05211 2 A1	<input checked="" type="checkbox"/>	Method and apparatus for developing software	717/100
25	US 20010 03744 8 A1	<input checked="" type="checkbox"/>	Input replicator for interrupts in a simultaneous and redundantly threaded processor	712/244
26	US 20010 03744 7 A1	<input checked="" type="checkbox"/>	Simultaneous and redundantly threaded processor branch outcome queue	712/239
27	US 20010 03625 5 A1	<input checked="" type="checkbox"/>	Methods and apparatus for providing speech recognition services to communication system users	379/88. 01
28	US 20010 03485 4 A1	<input checked="" type="checkbox"/>	Simultaneous and redundantly threaded processor uncached load address comparator and data value replication circuit	714/5
29	US 20010 03482 7 A1	<input checked="" type="checkbox"/>	Active load address buffer	712/225
30	US 20010 03482 4 A1	<input checked="" type="checkbox"/>	Simultaneous and redundantly threaded processor store instruction comparator	712/215
31	US 20010 02947 8 A1	<input checked="" type="checkbox"/>	System and method for supporting online auctions	705/37
32	US 20010 01689 9 A1	<input checked="" type="checkbox"/>	Data-processing device	712/215
33	US 20010 00588 0 A1	<input checked="" type="checkbox"/>	Information-processing device that executes general-purpose processing and transaction processing	712/34
34	US 67078 13 B1	<input checked="" type="checkbox"/>	Method of call control to minimize delays in launching multimedia or voice calls in a packet-switched radio telecommunications network	370/356
35	US 66679 88 B1	<input checked="" type="checkbox"/>	System and method for multi-level context switching in an electronic network	370/463

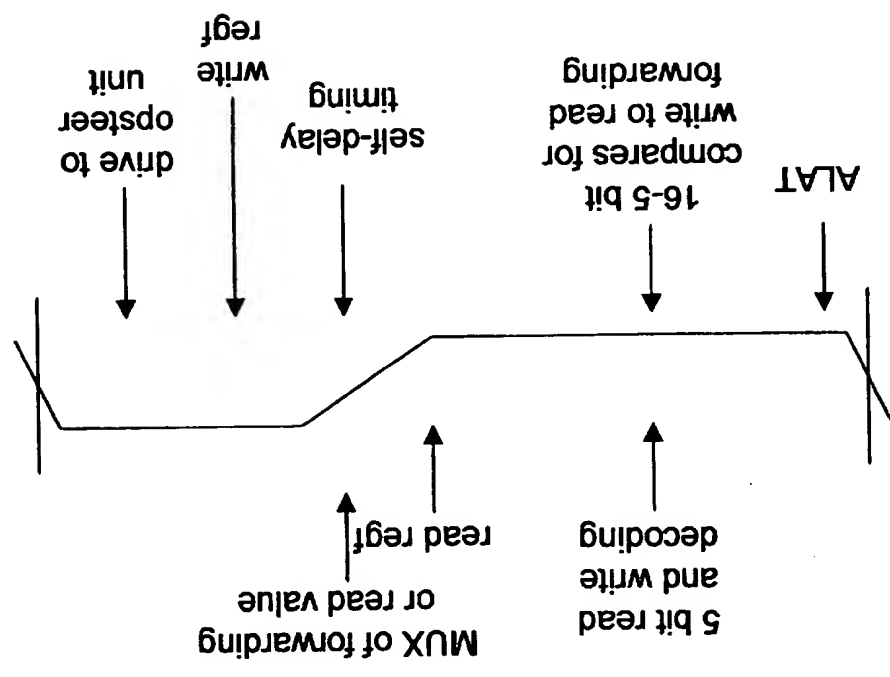


FIG. 37

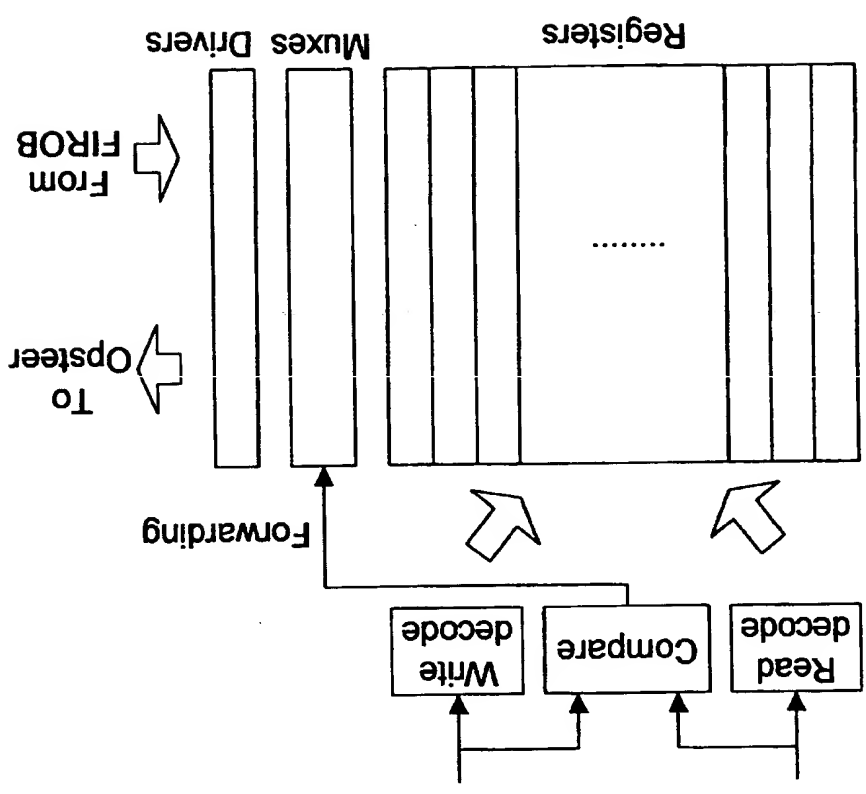


FIG. 38

	Docum ent ID	U	Title	Current OR
36	US 65981 51 B1	<input checked="" type="checkbox"/>	Stack Pointer Management	712/228
37	US 65981 22 B2	<input checked="" type="checkbox"/>	Active load address buffer	711/126
38	US 65565 63 B1	<input checked="" type="checkbox"/>	Intelligent voice bridging	370/352
39	US 65325 54 B1	<input checked="" type="checkbox"/>	Network event correlation system using formally specified models of protocol behavior	714/43
40	US 65300 80 B2	<input checked="" type="checkbox"/>	Method and apparatus for pre-processing and packaging class files	717/166
41	US 65075 92 B1	<input checked="" type="checkbox"/>	Apparatus and a method for two-way data communication	370/503
42	US 64700 81 B1	<input checked="" type="checkbox"/>	Telecommunications resource connection and operation using a service control point	379/221 .09
43	US 64184 60 B1	<input checked="" type="checkbox"/>	System and method for finding preempted threads in a multi-threaded application	718/108
44	US 64164 95 B1	<input checked="" type="checkbox"/>	Implantable fluid delivery device for basal and bolus delivery of medicinal fluids	604/132
45	US 63781 25 B1	<input checked="" type="checkbox"/>	Debugger thread identification points	717/129
46	US 63380 78 B1	<input checked="" type="checkbox"/>	System and method for sequencing packets for multiprocessor parallelization in a computer network system	718/102
47	US 62984 11 B1	<input checked="" type="checkbox"/>	Method and apparatus to share instruction images in a virtual cache	711/3
48	US 62333 15 B1	<input checked="" type="checkbox"/>	Methods and apparatus for increasing the utility and interoperability of peripheral devices in communications systems	379/88. 01
49	US 62298 80 B1	<input checked="" type="checkbox"/>	Methods and apparatus for efficiently providing a communication system with speech recognition capabilities	379/88. 01
50	US 62255 66 B1	<input checked="" type="checkbox"/>	Self-retaining screw spacer arrangement	174/138 E
51	US 61697 45 B1	<input checked="" type="checkbox"/>	System and method for multi-level context switching in an electronic network	370/463
52	US 61547 77 A	<input checked="" type="checkbox"/>	System for context-dependent name resolution	709/227
53	US 60785 64 A	<input checked="" type="checkbox"/>	System for improving data throughput of a TCP/IP network connection with slow return channel	370/235
54	US 59667 02 A	<input checked="" type="checkbox"/>	Method and apparatus for pre-processing and packaging class files	707/1
55	US 58729 63 A	<input checked="" type="checkbox"/>	Resumption of preempted non-privileged threads with no kernel intervention	712/233
56	US 58620 50 A	<input checked="" type="checkbox"/>	System for preparing production process flow	700/97
57	US 58127 60 A	<input checked="" type="checkbox"/>	Programmable byte wise MPEG systems layer parser	714/49
58	US 57614 92 A	<input checked="" type="checkbox"/>	Method and apparatus for uniform and efficient handling of multiple precise events in a processor by including event commands in the instruction set	712/244

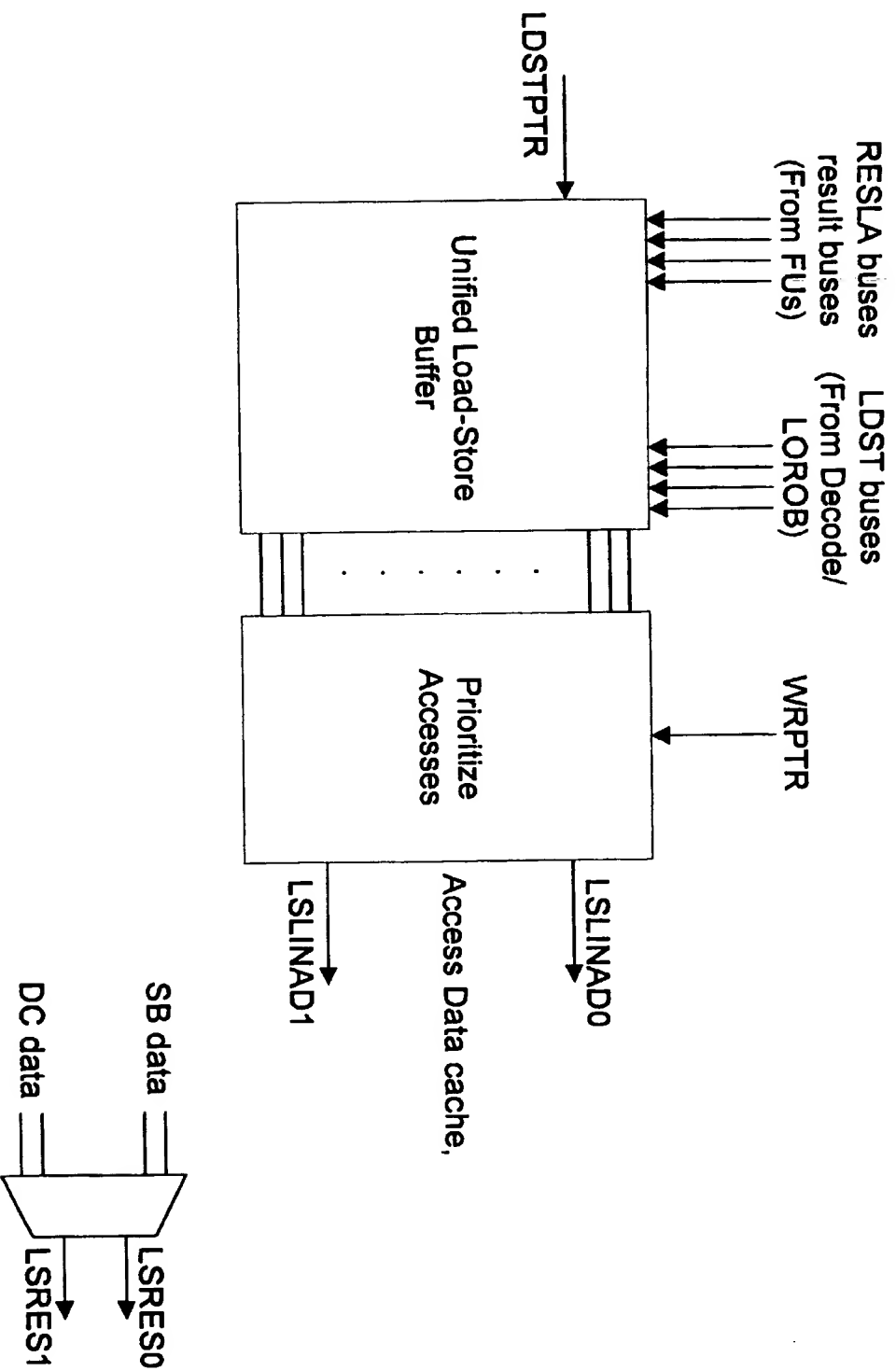
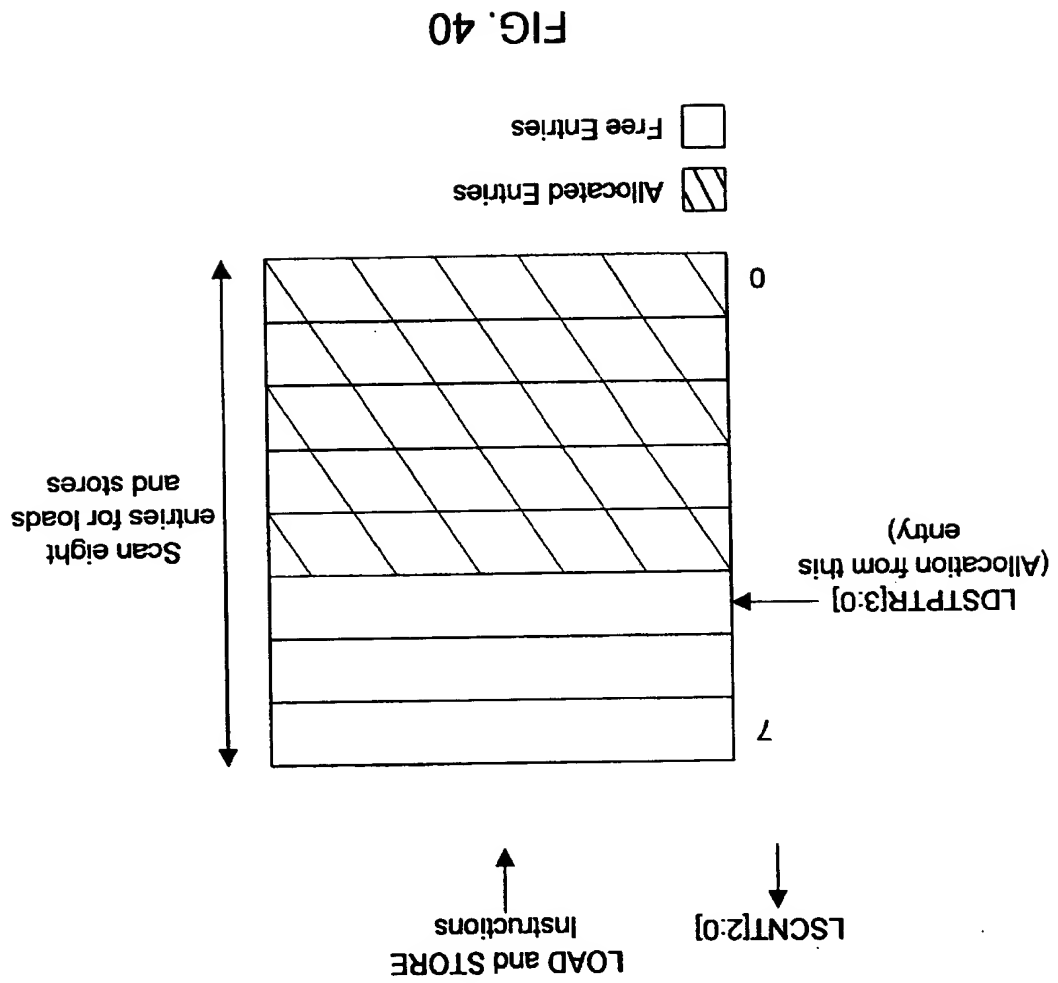


FIG. 39

	Docum ent ID	U	Title	Current OR
59	US 53575 08 A	<input checked="" type="checkbox"/>	Connectionless ATM network support using partial connections	370/397
60	US 52010 39 A	<input checked="" type="checkbox"/>	Multiple address-space data processor with addressable register and context switching	711/201
61	US 50086 58 A	<input checked="" type="checkbox"/>	Domed light housing for back-lit LCD display	345/87
62	US 47457 13 A	<input checked="" type="checkbox"/>	Prefabricated PC shelter structure	52/89
63	US 46316 05 A	<input checked="" type="checkbox"/>	Multiple speed scanner servo system for protecting the heads and tape of helical recorders	360/70



	Docum ent ID	U	Title	Current OR
1	JP 10152 810 A	<input type="checkbox"/>	CONNECTING CABLE	
2	JP 09137 424 A	<input checked="" type="checkbox"/>	INSTALLATION METHOD OF PC GIRDER	
3	JP 08249 183 A	<input checked="" type="checkbox"/>	EXECUTION FOR INFERENCE PARALLEL INSTRUCTION THREADS	
4	JP 03172 416 A	<input checked="" type="checkbox"/>	SHEATHING WALL STRUCTURE	
5	JP 02030 839 A	<input checked="" type="checkbox"/>	ANTICORROSIVE CONSTRUCTION IN ANCHOR SECTION OF PC STRAND	
6	WO 31076 16 A1	<input checked="" type="checkbox"/>	METHOD AND APPARATUS FOR INTERNET PROTOCOL HEADERS COMPRESSION INITIALIZATION	
7	WO 31071 07 A1	<input checked="" type="checkbox"/>	CONTROLLING AND/OR MONITORING DEVICE USING AT LEAST A TRANSMISSION CONTROLLER	
8	WO 99315 80 A1	<input checked="" type="checkbox"/>	PROCESSOR HAVING MULTIPLE PROGRAM COUNTERS AND TRACE BUFFERS OUTSIDE AN EXECUTION PIPELINE	
9	NNRD4 2676	<input checked="" type="checkbox"/>	Multiple Terminal Simulation Tool Utilizing Multiple Separate IP Addresses	
10	NNRD4 09126	<input checked="" type="checkbox"/>	Multiple Inline Multi-Function Tabs for Slider Controls	
11	NN910 5322	<input checked="" type="checkbox"/>	Determining Speaker-Dependent Phonetic Baseforms.	
12	NN880 5308	<input checked="" type="checkbox"/>	Card on Board Support System	
13	WO 20031 07107 A	<input checked="" type="checkbox"/>	Control and monitoring device for an automation system or similar, comprises peripherals linked to a central control unit that has two transmission controllers for controlling communications and setting up a security circuit	
14	JP 20032 68977 A	<input checked="" type="checkbox"/>	Connection tension device for connecting precast concrete steel materials, uses embedded bearing plate to support reaction force of connection cylinder which tightens steel materials	
15	US 20030 00526 2 A	<input checked="" type="checkbox"/>	Multithreaded processor for providing high instruction fetch bandwidth, has instruction buffer and temporary instruction cache to respectively receive different blocks of cache line	
16	JP 20021 41931 A	<input checked="" type="checkbox"/>	Router device for internet, has context ID rewriting unit which modifies similar context ID owned by several packets using unique number assigned to each address	
17	WO 20011 6718 A	<input checked="" type="checkbox"/>	Micro controlled function execution unit has controller maintaining multiple program counters and having logic for decoding instructions and context event arbiter to determine executable threads	
18	KR 20010 02485 A	<input checked="" type="checkbox"/>	Multi-thread microprocessor for instruction fetch	
19	US 61822 10 B	<input checked="" type="checkbox"/>	Processor with multiple program counters and trace buffers outside execution pipeline	
20	JP 08194 612 A	<input checked="" type="checkbox"/>	Program execution control for computer system - by calling second program counter according to analysis result of control data acquired after first program counter calls second program counter from control data gp.	
21	EP 52802 4 B	<input checked="" type="checkbox"/>	Error reporting for translated code execution - using address correlation table to identify source information in original code from error address in translated code	
22	GB 20967 19 A	<input checked="" type="checkbox"/>	Clutch arrangement for marine propulsion drive - includes engageable pawl detent causing screwing of bush to engage intermediate clutch part, when drive side speed drops	

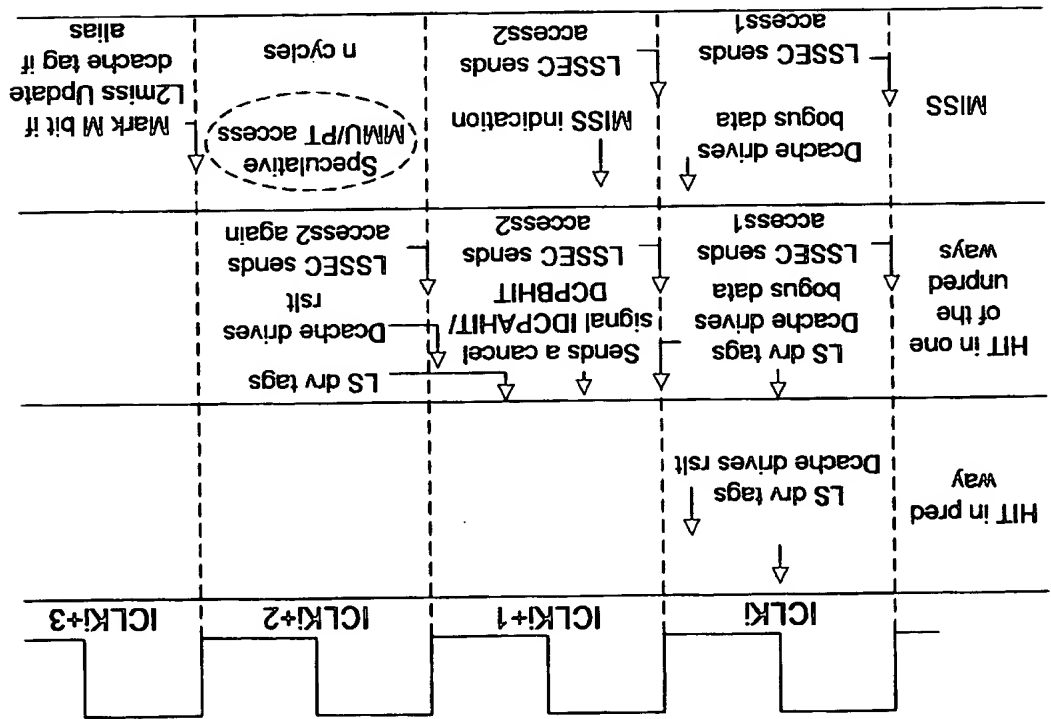


FIG. 43

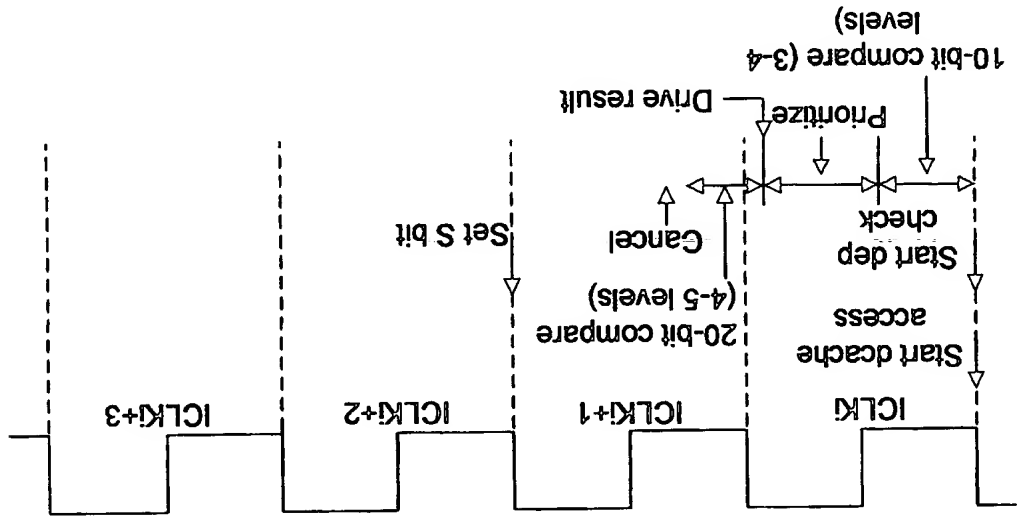
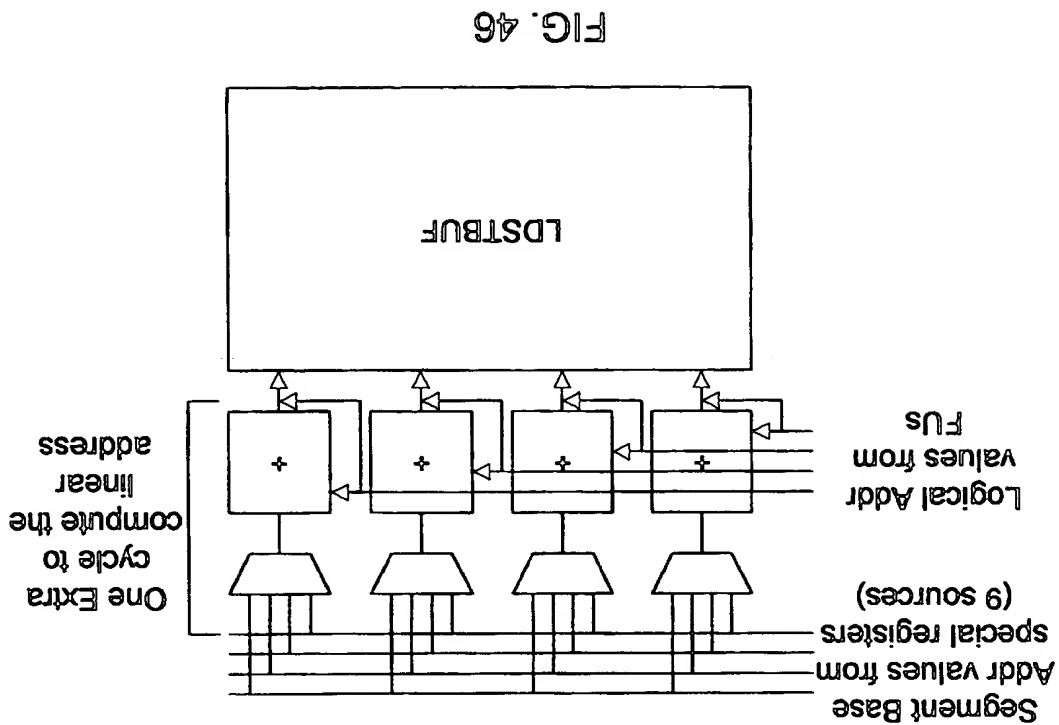
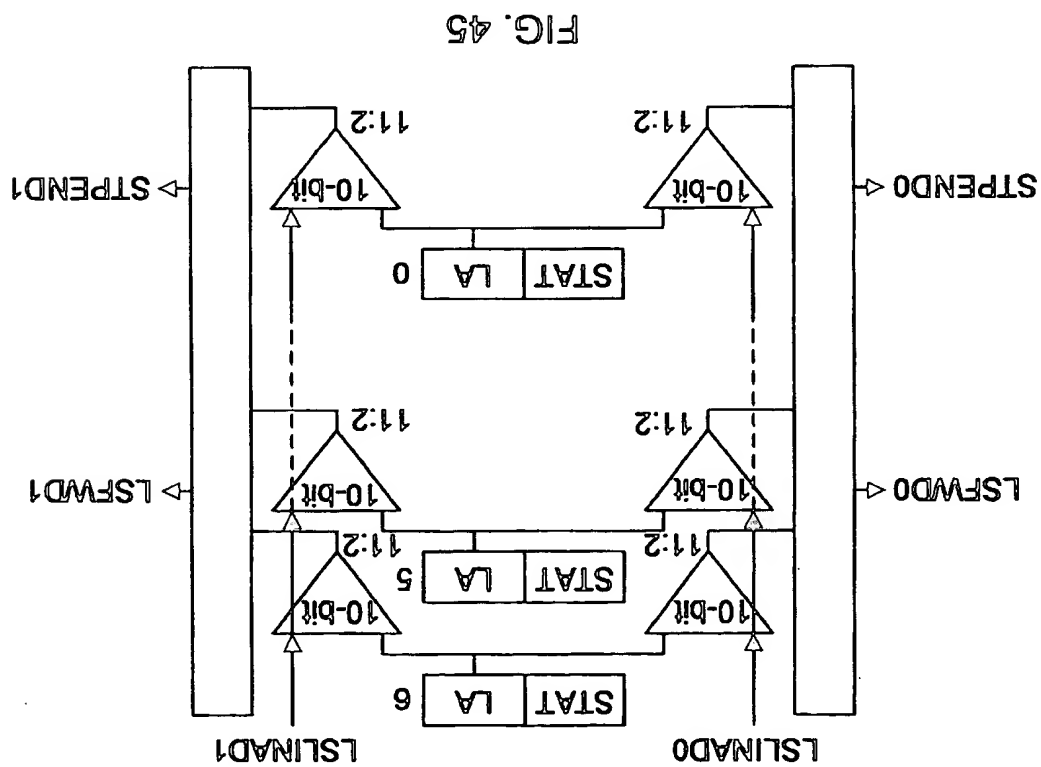
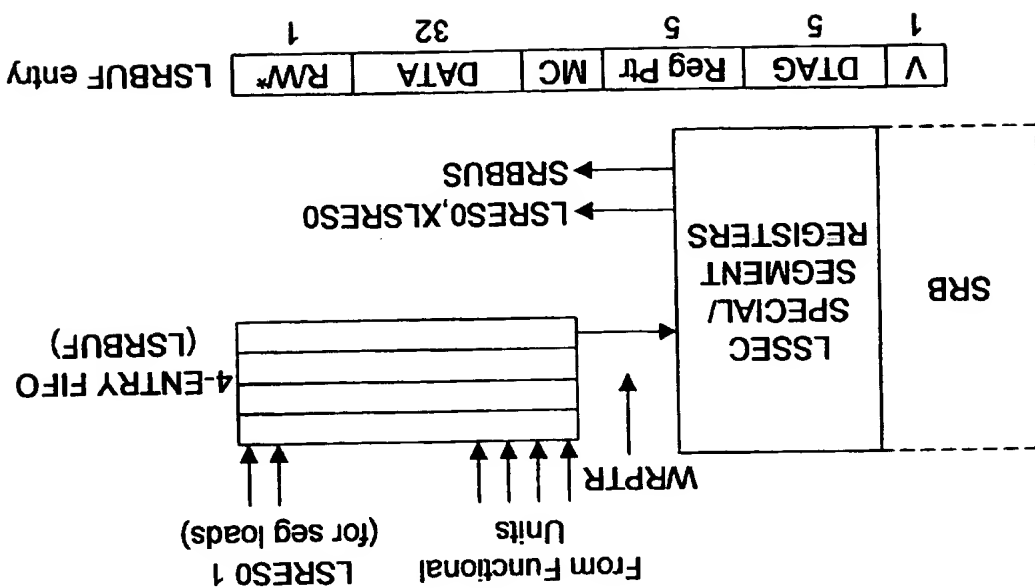
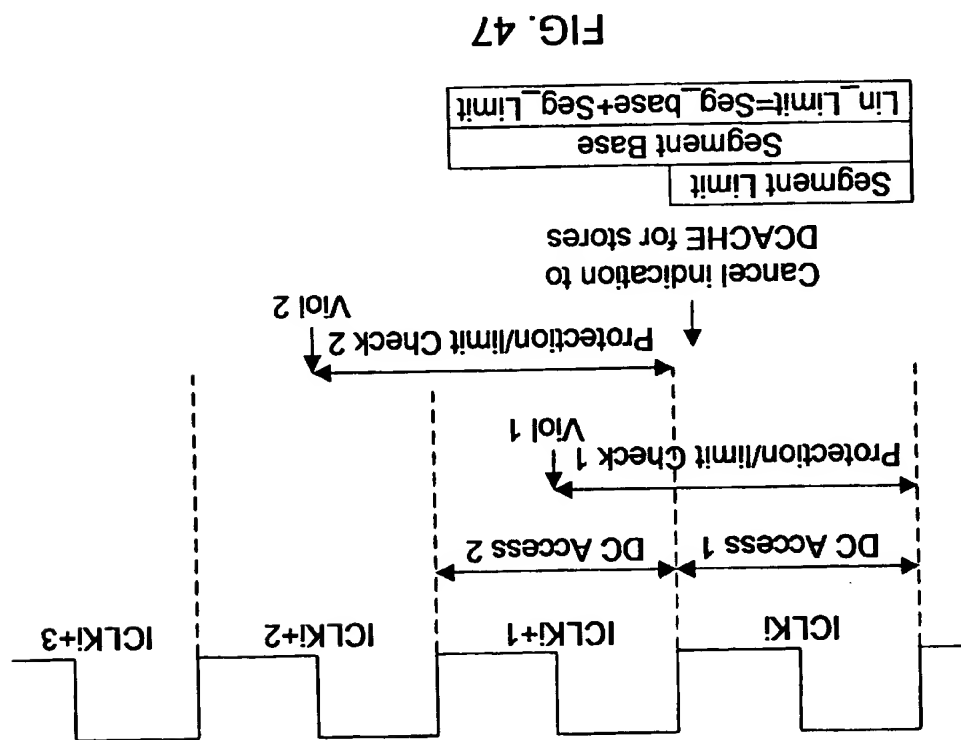


FIG. 44

	Docum ent ID	U	Title	Current OR
1	JP 20013 06263 A	<input type="checkbox"/>	MEDIA DATA STORAGE DEVICE	
2	JP 20011 42937 A	<input checked="" type="checkbox"/>	SCHEDULING CORRECTNESS CHECKING METHOD AND SCHEDULE VERIFYING METHOD FOR CIRCUIT	
3	JP 20010 46747 A	<input checked="" type="checkbox"/>	PROCESSING CONTROL METHOD FOR VIDEO GAME, RECORDING MEDIUM RECORDING PROCESS CONTROL PROGRAM, AND GAME DEVICE	
4	JP 20003 11253 A	<input checked="" type="checkbox"/>	THREE-DIMENSIONAL IMAGE GENERATION SYSTEM AND METHOD AND RECORDING MEDIUM	
5	JP 11296 385 A	<input checked="" type="checkbox"/>	CONTEXT CONTROLLER FOR MANAGING MULTI-TASKING BY PROCESSOR	
6	JP 11272 484 A	<input checked="" type="checkbox"/>	DEVICE AND METHOD FOR DISPATCHING CLIENT REQUEST	
7	JP 11023 420 A	<input checked="" type="checkbox"/>	METHOD FOR EVALUATING LIFE OF BOLT	
8	JP 09026 923 A	<input checked="" type="checkbox"/>	METHOD AND DEVICE FOR MANAGING OBJECT AND PROCESS INSIDE DISTRIBUTED OBJECT OPERATING ENVIRONMENT	
9	JP 08153 023 A	<input checked="" type="checkbox"/>	INSTRUCTION EXECUTION FREQUENCY MEASURING METHOD AND MULTIPROCESSOR SYSTEM USING THE METHOD	
10	JP 07000 667 A	<input checked="" type="checkbox"/>	THREAD CUT CONTROL DEVICE FOR SEWING MACHINE	
11	JP 02177 995 A	<input checked="" type="checkbox"/>	BUTTON HOLDING MECHANISM FOR BUTTON SEWING MACHINE	
12	JP 01292 430 A	<input checked="" type="checkbox"/>	PARALLEL PROCESSING PROCESSOR	
13	JP 01274 240 A	<input checked="" type="checkbox"/>	PARALLEL PROCESSING PROCESSOR	
14	WO 22529 A1	<input checked="" type="checkbox"/>	METHOD FOR PROCESSING AN ELECTRONIC SYSTEM SUBJECTED TO TRANSIENT ERROR CONSTRAINTS AND MEMORY ACCESS MONITORING DEVICE	
15	EP 94236 6 A2	<input checked="" type="checkbox"/>	Event-driven and cyclic context controller and processor employing the same	
16	EP 94236 5 A2	<input checked="" type="checkbox"/>	Context controller having instruction-based time slice task switching capability and processor employing the same	
17	EP 94236 4 A2	<input checked="" type="checkbox"/>	Context controller having status-based background task resource allocation capability and processor employing the same	
18	EP 73547 5 A2	<input checked="" type="checkbox"/>	Method and apparatus for managing objects in a distributed object operating environment	
19	DE 43318 67 A1	<input checked="" type="checkbox"/>	Method and winding device for producing wound objects, especially fruit gum whirls	
20	DE 43149 82 A1	<input checked="" type="checkbox"/>	Method for making a thread connection by splicing	
21	EP 61736 1 A2	<input checked="" type="checkbox"/>	Scheduling method and apparatus for a communication network.	



	Document ID	U	Title	Current OR
22	EP 567428 A1	<input checked="" type="checkbox"/>	Process for starting a loom and loom for effecting the same.	
23	GB 2234613 A	<input checked="" type="checkbox"/>	Method and apparatus for switching contexts in a microprocessor	
24	DE 3433027 A1	<input checked="" type="checkbox"/>	Lockstitch sewing machine	
25	EP 126035 A1	<input checked="" type="checkbox"/>	Device for producing knitted pockets with the aid of a circular machine for products manufactured in a tubular way with continuous unidirectional movement, and machine equipped for this device.	
26	NN9211392	<input checked="" type="checkbox"/>	User Interface Architecture for Response Time Dependent Control.	
27	US 20040003023 A	<input checked="" type="checkbox"/>	Computer system processor selection method for loading processing thread, involves overwriting candidate/volunteer processor information based on comparison of load between candidate/volunteer processor	
28	US 6640299 B	<input checked="" type="checkbox"/>	Computation engine access arbitrating method for video graphics controller, involves determining priority operation instruction based on application specification prioritization scheme when multiple codes are pending	
29	US 20030097548 A	<input checked="" type="checkbox"/>	Processing system e.g. pipelined computer processing system, selects address and instructions from respective context registers simultaneously for each cycle of execution of processor unit	
30	US 20030046517 A	<input checked="" type="checkbox"/>	Multithreading apparatus for computer processor pipeline, has control unit statically scheduled to execute multiple threads in round robin succession to eliminate need for communication between pipeline stages	
31	US 20020103990 A	<input checked="" type="checkbox"/>	Multithreaded processor architecture e.g. for precession computer, includes cycle allocation table containing thread identifiers for active threads, and different execution time allotted to each thread	
32	US 20020069345 A	<input checked="" type="checkbox"/>	Very long instruction word processor includes threads comprising processing units which execute respective issue groups of VLIW packets in single clock cycle	
33	US 20020002667 A	<input checked="" type="checkbox"/>	Embedded processor architecture for enabling multithreading, invokes zero-time context switches between end and beginning of program instruction execution states in threads	
34	JP 2001142937 A	<input checked="" type="checkbox"/>	Scheduling correctness checking method for circuit, involves executing symbolic simulation for extracting loop invariant term for determining sufficient set of non-cyclic thread	
35	US 6341347 B	<input checked="" type="checkbox"/>	Multiple thread processor has thread switch logic for switching execution threads according to thread switching mode selected from multiple thread switching modes	
36	WO 200052244 A	<input checked="" type="checkbox"/>	Production of tubular knitwear such as bras, panties, involves producing specific lengths of tubular fabric having cylindrical shape by excluding predetermined number of needles of needlebed	
37	US 6466898 B	<input checked="" type="checkbox"/>	Event driven logic simulation executing method for design and debug of VLSI circuit, involves creating master and slave threads for execution of logic simulation algorithm on processor platform	
38	US 20020092579 A	<input checked="" type="checkbox"/>	Weaving machine for kelim and gobelin fabrics	
39	BE 1011262 A	<input checked="" type="checkbox"/>	Double plush carpet weaving method - involves interlacing pile threads with two base cloths in at least three different patterns to form double-plush and flat weave regions	
40	KR 98063489 A	<input checked="" type="checkbox"/>	Multithread data processing system for RISC architecture, has storage control unit for extracting requested data and instruction for execution and instruction units based on data and instruction fetch request	



	Docum ent ID	U	Title	Current OR
41	EP 89874 3 B	<input checked="" type="checkbox"/>	Multi-thread microprocessor for executing interrupt service routines as thread - includes context file which stores multiple contexts each associated with thread, such that multiple threads may be concurrently executed	
42	EP 72403 2 A	<input checked="" type="checkbox"/>	Weaving machine solenoid controlled thread selecting device - includes thread selecting units each comprising hook pivoting between selection needle engaging and release positions according to selective energised state of solenoid	
43	US 54902 72 A	<input checked="" type="checkbox"/>	Creating multi-threaded processing cycle time slices for application program running on 32-bit desk-top multi-tasking operating system - obtaining 1st time slice for thread having fixed execution time, loading threadlet control block and executing threadlets for threadlet control block in order specified and within fixed execution time of time slice	
44	EP 57094 7 A	<input checked="" type="checkbox"/>	Jacquard pulley tackle - has structured layout of two rollers and roller unit with cord hooks to reduce assembly height	
45	EP 53601 0 A	<input checked="" type="checkbox"/>	Management in real-time of multi-function processor - using time slicing to move between processes with context saving at end of time slice to allow fast context restoration	
46	DE 40205 50 C	<input checked="" type="checkbox"/>	Warp knitting machine with needle bed with guide bars - provides knitted goods with stable surface	
47	GB 22346 13 A	<input checked="" type="checkbox"/>	Fast microprocessor, switching contexts when cache miss occurs - couples copies of state elements to multiplexer to save context of instructions and resume execution in one cycle	
48	EP 38347 4 A	<input checked="" type="checkbox"/>	Programmed peripheral controller - uses cyclic reassessment of existing and pending contexts and priorities to select correct context to execute	
49	EP 36460 0 B	<input checked="" type="checkbox"/>	Round screw machining method - executing repetition of screw threading cycles obtained by NC tool data, while shifting sequentially cycle start point along arc shape	
50	EP 29547 0 A	<input checked="" type="checkbox"/>	Call transfer procedure in computer controlled exchange - first sending unanswered ringing current to selected group of phones before extending to all	
51	JP 63275 389 A	<input checked="" type="checkbox"/>	Appts. to detect remaining amt. of bobbin thread in sewing machine - contg. bobbin thread remaining amt. counters for different types of bobbins, and selector	
52	JP 86042 590 B	<input checked="" type="checkbox"/>	Control unit for automatic sewing machine - in which mode is selected for next cycle when abnormal condition is detected (J5 6.5.82)	
53	DE 35866 03 G	<input checked="" type="checkbox"/>	Threaded interpretive language data processor - effects multiple data selection and arithmetic operations in single clock cycle using parameter stack implemented in hardware	
54	EP 12603 5 A	<input checked="" type="checkbox"/>	Tubular knitting machine with rotary needle selection device - for producing hose with integral heels, ends, gussets etc.	
55	BE 90007 5 A	<input checked="" type="checkbox"/>	Weft yarn spool drives for shuttle free loom - with stepping drive electric motors under programmed pulse control with angular references	
56	EP 11415 3 A	<input checked="" type="checkbox"/>	Circular profile knitting machine with variable thread feed rate - to eliminate residual tension in yarns forming stocking heels or other profile variations	
57	SU 10965 29 A	<input checked="" type="checkbox"/>	Cycling bend and twist test for flexible components - with reciprocating bending surfaces linked with torsion oscillation of end grip	
58	EP 80581 A	<input checked="" type="checkbox"/>	Loom control system - applies additional warp tension at re-start, varying according to loom angle sensed by detector	
59	BE 89338 9 A	<input checked="" type="checkbox"/>	Weft supply elements, selected by electromagnets - move from wait to transfer positions against spring restraint, for shed insertion	
60	SU 50412 8 A	<input checked="" type="checkbox"/>	Textile threads flexure destructive testing - uses complex waveform vibrator and two clamps with fixing jaws	
61	DE 19215 76 A	<input checked="" type="checkbox"/>	Actuating weft threads during colour change in - shuttless weaving looms	

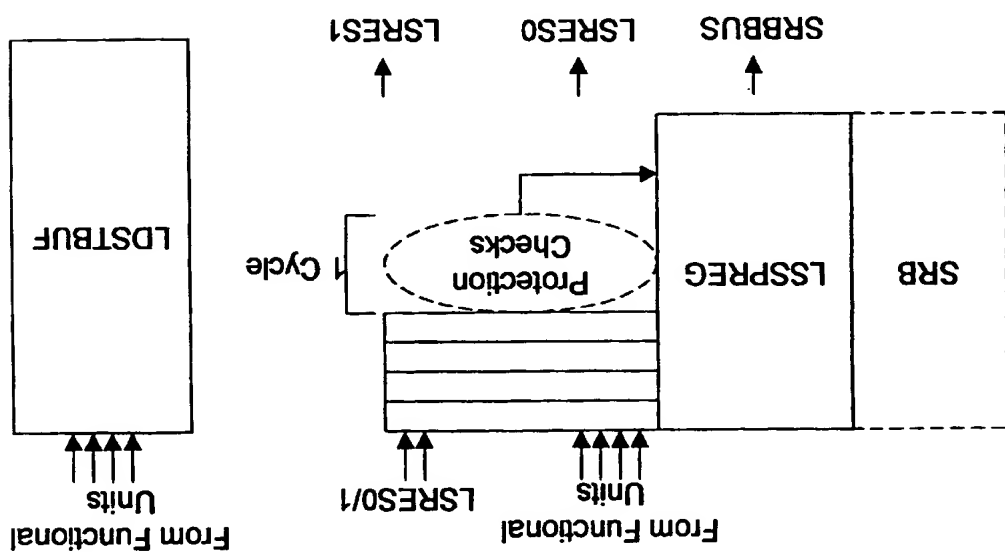


FIG. 49

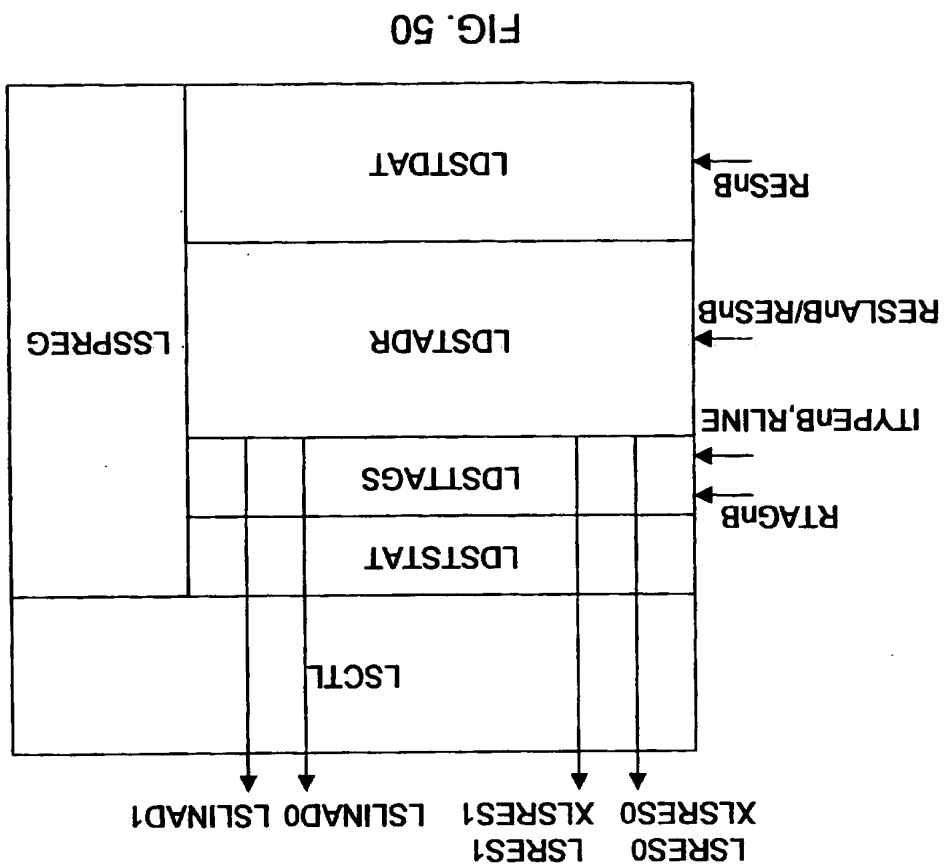


FIG. 50

	L #	Hits	Search Text	DBs
1	L1	683	(select\$3 interleav\$3 execut\$3) near10 ((cycl\$6 robin) near20 (multithread\$3 thread\$3 context))	USPAT; US-PGPUB
2	L2	15106	select\$3 near10 (pc pa ip ((instruction program) adj2 (counter address)))	USPAT; US-PGPUB
3	L4	58	2 near99 (multithread\$3 thread\$3 context) and pipelin\$3	USPAT; US-PGPUB
4	L5	327	1 and pipelin\$3	USPAT; US-PGPUB
5	L6	189	(multithread\$3 thread\$3 context).ab,ti. and 5	USPAT; US-PGPUB

ICSTORE
 Processor 500 executes fast X86 instructions directly, no ROPs are needed. The pre-decode bits with each byte of instruction are 3 bits; start bit, end bit, and functional bit. All the externally fetched instructions will be latched into the ICACHE. This should not be a problem since the ICACHE is idle and waits for external instructions. Only single byte prefix of 0x66 and 0x0F is allowed for Processor 500's fast path, multiple prefixes including 0x67 is allowed for multi-prefix, all other prefixes will take an extra cycle in decoding or go to MROM. With these simple prefixes, the instruction bytes need not be modified. The linear valid bit is used for the whole cache-line of instructions, 16-byte. The replacement procedure is done by the CMASTER. Along with each line of instruction, the CMASTER tells the ICACHE which way to put in the data and tag. The start and end bits are sufficient to validate the instruction. If branching to the middle of the line or instructions wrapping to the next line, the start and this must be detected for each instruction or else the instruction must be pre-decoded again. The possible cases are branching to the opcode and skipping the prefix (punning of instruction) and part of the wrapping instruction is replaced in the ICACHE. The instructions must first pass through the pre-fetch buffers before sending to the ICACHED. The ICACHED has only one input from the IB(127:0) for both the pre-decode or cached instructions. The pre-decode information is written into the ICPCDAT as the whole line is decoded. The output IB(127:0) is merged with the previous 8-byte to form a 24-byte line for the alignment unit to select and send to 4 decode units.

Since the instruction fetching from external memory will be written directly into the ICACHE, the pre-fetch buffer should be built into the ICSTORE; the input/output path of the array. In this way, the data will be written into the ICACHE regardless of the pre-decode information or the taken branch instruction and the instructions is available to the ICACHE as soon as they are valid on the bus. The number of pre-fetch buffers is two, and request will be made to BIU as soon as there is space in the pre-fetch buffer for another line of instructions. The pre-fetch buffer consists of a counter and a valid bit for instructions written into the cache and a valid bit for instructions sent to the decode unit. As long as the address pointer is still in the same block, the data will be written to the array. With the pre-fetch buffer in the ICACHE, a dedicated bus should be used to transfer instructions directly from the pads to the ICACHE; this is a step to keep Processor 500 from using dynamic pre-charged buses.

ICSTORE ORGANIZATION
 The ICSTORE on Processor 500 does not store the pre-decode data, as shown in FIG. 9. The ICSTORE consists of 32K bytes of instructions organized as 8 sets of 128 rows by 256 columns. The array set in this documentation has its own decoder. The decoder is in the center of the set. Each of the sets consist of 2-byte of instructions. The 8-way associative muxing from the 8 TAG-HITS is performed before the data is routed to the ICALIGN block. With this arrangement, the input/output to each set is 16-bit buses. The decode unit should also be decoded; this topic will be discussed in detail below in the ICALIGN block section. For optimal performance the layout of the column should be 64 RAM cells, pre-charge, 64 RAM cells, write buffer and senseamp. The row decoder should be in the middle of the array to drive 128 column each way. Basically, the pre-charge and the row decoder should be crossed in the middle of the array. The self-time column is used to generate internal clock for each set of the array. Pre-charge is gated

RETPRED—Output to Idecode indicates the current prediction of the return instruction of the fetched line. The return instruction must be detected in the current line of instruction or the ICACHE must be re-fetched from a new line. IC_EXT_RD—Output to CMASTER indicates the next line of instruction should be fetched from external register-less of aliasing. This is for pre-fetching of instruction which crosses the line boundary. MMUPFFGFT—Input from MMU indicates page fault for the current instruction address. TLB_MISS_PFI—Input from MMU indicates TLB miss for the current instruction address. PFI_XFER—Output to CMASTER indicates the address for the current line is written into the cache, the L2 should be updated with the physical address. This is 15 of instruction, the CMASTER tells the ICACHE which way to put in the data and tag. The start and end bits are sufficient to validate the instruction. If branching to the middle of the line or instructions wrapping to the next line, the start and this must be detected for each instruction or else the instruction must be pre-decoded again. The possible cases are branching to the opcode and skipping the prefix (punning of instruction) and part of the wrapping instruction is replaced in the ICACHE. The instructions must first pass through the pre-fetch buffers before sending to the ICACHED. The ICACHED has only one input from the IB(127:0) for both the pre-decode or cached instructions. The pre-decode information is written into the ICPCDAT as the whole line is decoded. The output IB(127:0) is merged with the previous 8-byte to form a 24-byte line for the alignment unit to select and send to 4 decode units.

BIT0MASK—Input from CMASTER indicates masking of bit 20 for backward compatible with 8086. The line should not be cache if outside of the page. BSTRUN—Input from TAP indicates to start the BIST. BSTRD—Input from TAP indicates to read the array and compare to set the result. BSTRW—Input from TAP indicates to write the array from input registers. BSTRST—Input from TAP indicates to reset the counter. BSTRINC—Input from TAP indicates to increment the counter. BSTDIN—Input from TAP indicates the test pattern to the input registers. The input can be from the TDI pin or normal burn-in patterns. FLUSHON—Input from TAP indicates flushing register compare input for flushing the result registers. UPDOWN—Input from TAP indicates counting up or down. BSTRHI—Input from TAP indicates shifting of the master latch of registers. BSTRHI2—Input from TAP indicates shifting of the slave latch of registers. BSTRALSE—Input from TAP indicates to invert the test pattern. PORTSEL—Input from TAP indicates to select the second dual port. BSTDOUT—Output to TAP indicates the result of the data chain from the ICSTORE and ICPCDAT arrays. BSTDOUT2—Output to TAP indicates the result of the data chain from the ICNXTBLK and ICTAGV arrays. BSTMASB—Output to TAP indicates maximum count for dual port arrays. MAXADDR—Output to TAP indicates maximum index counter. ATPGIN(15:14)—Input from dedicated pins for ATPG. ATPGOUT(15:14)—Output to dedicated pins for ATPG.

	Docum ent ID	U	Title	Current OR
1	US 20040 07391 0 A1	<input type="checkbox"/>	Method and apparatus for high speed cross-thread interrupts in a multithreaded processor	719/310
2	US 20040 07390 5 A1	<input checked="" type="checkbox"/>	Method and apparatus to quiesce a portion of a simultaneous multithreaded central processing unit	718/101
3	US 20040 07378 1 A1	<input checked="" type="checkbox"/>	Method and apparatus for token triggered multithreading	712/235
4	US 20040 07377 9 A1	<input checked="" type="checkbox"/>	Method and apparatus for register file port reduction in a multithreaded processor	712/225
5	US 20040 07377 8 A1	<input checked="" type="checkbox"/>	Parallel processor architecture	712/220
6	US 20040 07377 2 A1	<input checked="" type="checkbox"/>	Method and apparatus for thread-based memory access in a multithreaded processor	712/1
7	US 20040 07052 6 A1	<input checked="" type="checkbox"/>	ARITHMETIC DECODING METHOD AND AN ARITHMETIC DECODING APPARATUS	341/107
8	US 20040 05499 0 A1	<input checked="" type="checkbox"/>	Post-pass binary adaptation for software-based speculative precomputation	717/124
9	US 20040 05488 0 A1	<input checked="" type="checkbox"/>	Microengine for parallel processor architecture	712/245
10	US 20040 03485 8 A1	<input checked="" type="checkbox"/>	Programming a multi-threaded processor	718/108
11	US 20040 03475 9 A1	<input checked="" type="checkbox"/>	Multi-threaded pipeline with context issue rules	712/1
12	US 20040 00658 4 A1	<input checked="" type="checkbox"/>	Array of parallel programmable processing engines and deterministic method of operating the same	718/107
13	US 20030 23339 4 A1	<input checked="" type="checkbox"/>	Method and apparatus for ensuring fairness and forward progress when executing multiple threads of execution	718/107
14	US 20030 22597 5 A1	<input checked="" type="checkbox"/>	Method and apparatus for multithreaded cache with cache eviction based on thread identifier	711/133
15	US 20030 21288 1 A1	<input checked="" type="checkbox"/>	Method and apparatus to enhance performance in a multi-threaded microprocessor with predication	712/226
16	US 20030 20042 4 A1	<input checked="" type="checkbox"/>	Master-slave latch circuit for multithreaded processing	712/228
17	US 20030 19192 7 A1	<input checked="" type="checkbox"/>	Multiple-thread processor with in-pipeline, thread selectable storage	712/228

T

1. Field of the Invention

2. Description of the Relevant Art

Superscalar microprocessors are

...the shortest possible clock cycle consistent with

interval of time in which the various stages of the

elements (such as registers or arrays) according to a clock

limb may capture a value occurring along edge of the clock signal.

their microprocessors in accordance with the x86 multiprocessor architecture. The x86 microprocessor architecture is

Large body of software exists which runs only on micropro-

consistently retain compatibility with this body of software. As

useful for passing information between a program and a

subroutine performs a function that a program requires, and then returns to the instruction following the call to the

multiple places within a program to perform its function. A

Of the instruction subsequent to the call instruction is saved

address". The instruction is referred to as an "address". The instruction which causes program execution

In the x86 architecture, the ESP register points to the

structure in which values are placed on the stack in a certain

on the stack. The action of placing a value on the stack is

the stack is referred to as a "pop", and requesting that a po

is performed, the ESP register is decremented by the size of

mented ESP register value. When a pop command

are copied from the top of the stack to a register—by the pop command, and then the ESP register is incremented.

An example of the use of push and pop commands in a

microprocessor.

A branch instruction is an instruction which causes the next

addresses. One address is the address immediately following the branch instruction. This address is referred to as the

by the branch instruction, and is referred to as the branch "target address" or simply the "target address". Branch

the next sequential address based on a particular condition

instruction is not known until the branch instruction

executes, supervisor instructions either return to the branch instruction or continue to the next instruction (reducing

tion will select when executed. When the prediction method is chosen the resulting superscalar microprocessor may

predicted address. If the prediction is incorrect, or mispre-

non are discarded from the instruction processing pipeline and the correct instructions are fetched. Branch instructions

decoded or when instructions are fetched, depending on the

be considered to be branches which always select the target

dict in the x86 microprocessor architecture is the RET

processor architecture). The return instruction is a pop command, as described above. This type of branch instruc-

return address) is not readily available when the instruction is decoded unlike some other branch instructions. Instead,

to be indicated by the value in the ESP register when the return instruction is executed. The value of the ESP register at the

ESP register at the time the return instruction is executed

well.

normally used in conjunction with the CALL instruction.

immediately following the call to be pushed onto the stack,

by the CALL instruction (i.e. the CALL instruction is the

command, as described above, as well as a branch instruc-

	Document ID	U	Title	Current OR
18	US 20030 19186 6 A1	<input checked="" type="checkbox"/>	Registers for data transfers	719/313
19	US 20030 18814 1 A1	<input checked="" type="checkbox"/>	Time-multiplexed speculative multi-threading to support single-threaded applications	712/235
20	US 20030 18234 6 A1	<input checked="" type="checkbox"/>	Method and apparatus for configuring arbitrary sized data paths comprising multiple context processing elements	708/700
21	US 20030 17225 6 A1	<input checked="" type="checkbox"/>	Use sense urgency to continue with other heuristics to determine switch events in a temporal multithreaded CPU	712/228
22	US 20030 16367 5 A1	<input checked="" type="checkbox"/>	Context switching system for a multi-thread execution pipeline loop and method of operation thereof	712/228
23	US 20030 16366 9 A1	<input checked="" type="checkbox"/>	Configuration of multi-cluster processor from single wide thread to two half-width threads	712/24
24	US 20030 16366 8 A1	<input checked="" type="checkbox"/>	Local control of multiple context processing elements with configuration contexts	712/15
25	US 20030 15888 5 A1	<input checked="" type="checkbox"/>	Method and apparatus for controlling the processing priority between multiple threads in a multithreaded processor	718/108
26	US 20030 15423 5 A1	<input checked="" type="checkbox"/>	Method and apparatus for controlling the processing priority between multiple threads in a multithreaded processor	718/108
27	US 20030 14996 4 A1	<input checked="" type="checkbox"/>	Method of executing an interpreter program	717/138
28	US 20030 14517 3 A1	<input checked="" type="checkbox"/>	Context pipelines	711/140
29	US 20030 13571 6 A1	<input checked="" type="checkbox"/>	Method of creating a high performance virtual multiprocessor by adding a new dimension to a processor's pipeline	712/220
30	US 20030 13571 1 A1	<input checked="" type="checkbox"/>	Apparatus and method for scheduling threads in multi-threading processors	712/200
31	US 20030 12640 3 A1	<input checked="" type="checkbox"/>	Method and apparatus for retiming in a network of multiple context processing elements	712/11
32	US 20030 12089 6 A1	<input checked="" type="checkbox"/>	System on chip architecture	712/32
33	US 20030 10594 4 A1	<input checked="" type="checkbox"/>	Method and apparatus to quiesce a portion of a simultaneous multithreaded central processing unit	712/220
34	US 20030 10590 1 A1	<input checked="" type="checkbox"/>	PARALLEL MULTI-THREADED PROCESSING	710/240

sponding to previously fetched call instructions. The main memory is configured to store instructions for execution by the microprocessor and further configured to store data for manipulation by the microprocessor.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

FIG. 1 is a block diagram of one embodiment of a superscalar microprocessor including an instruction cache and a branch prediction unit.

FIG. 2 is a block diagram of one embodiment of a return prediction unit which may be included within the instruction cache or the branch prediction unit shown in FIG. 1.

FIG. 2A is a diagram of another embodiment of a return stack storage which may be included in the return prediction unit shown in FIG. 2.

FIG. 3 is a diagram of instructions, including call and return instructions, illustrating instruction flow according to call and return instructions.

FIG. 4A is an exemplary instruction stream used to illustrate the function of the present return prediction unit.

FIG. 4B shows the contents of the return address storage shown in FIG. 2 prior to executing the instruction stream shown in FIG. 4A.

FIG. 4C shows the contents of the return address storage shown in FIG. 2 after the execution of several instructions from the exemplary instruction stream shown in FIG. 4A.

FIG. 4D shows the contents of the return address storage shown in FIG. 2 after the execution of several more instructions from the exemplary instruction stream shown in FIG. 4A.

FIG. 4E shows the contents of the return address storage shown in FIG. 2 after completing execution of the exemplary instruction stream shown in FIG. 4A.

DETAILED DESCRIPTION OF THE INVENTION

Turning now to FIG. 1, a block diagram of one embodiment of a superscalar microprocessor 200 in accordance with the present invention is shown. As illustrated in the embodiment of FIG. 1, superscalar microprocessor 200 includes a prefetch/predcode unit 202 and a branch prediction unit 220 coupled to an instruction cache 204. An instruction alignment unit 206 is coupled between instruction cache 204 and a plurality of decode units 208A-208D (referred to collectively as decode units 208). Each decode unit 208A-208D is coupled to respective reservation station units 210A-210D (referred to collectively as reservation stations 210), and each reservation station 210A-210D is coupled to a respective functional unit 212A-212D (referred

to as CALL instruction address placed on the stack by the CALL instruction is the address intended to be used by the instruction as the return address. The CALL instruction can therefore be used to call a subroutine in a program, and the subroutine typically ends with a return instruction which causes instruction execution to resume at the instruction immediately following the CALL instruction.

"Fake return" instructions are return instructions which are executed when a return address other than a return address provided by a CALL instruction is at the top of the stack. This address may be placed on the stack by executing a PUSH instruction, for example. A mechanism for predicting the target address of a return instruction is desired.

SUMMARY OF THE INVENTION

The problems outlined above are in large part solved by a microprocessor in accordance with the present invention. The present microprocessor employs a return prediction unit configured to predict return addresses for return instructions according to a return stack storage included therein. The return stack storage is a stack structure configured to store return addresses associated with previously detected call instructions. Advantageously, return addresses may be predicted for return instructions early in the instruction processing pipeline of the microprocessor. Instructions residing at the target of the return instruction may be fetched more quickly than was previously achievable using conventional superscalar microprocessors. Performance may be increased according to the decreased time between execution of the return instruction and execution of the instructions stored at the return address.

In one embodiment, the return stack storage additionally stores a call tag and a return tag with each return address. The call tag and return tag respectively identify call and return instructions associated with the return address. These tags may be compared to a branch tag conveyed to the return prediction unit upon detection of a branch misprediction. The results of the comparisons may be used to adjust the contents of the return stack storage with respect to the misprediction. Advantageously, the return prediction unit recovers from mispredicted branches. In other words, the return prediction unit may continue to predict return addresses correctly following a mispredicted branch instruction. Because mispredicted branch recovery is often an important feature of superscalar microprocessors, performance may be increased by recovering the return prediction unit. Broadly speaking, the present invention contemplates a superscalar microprocessor comprising a branch prediction unit and a return prediction unit. The branch prediction unit is configured to predict branch instructions taken or not taken, wherein branch instructions include call instructions and return instructions. Coupled to receive an indication of the call instructions and the return instructions from the branch prediction unit, the return prediction unit is configured to predict branch instructions taken or not taken, wherein branch instructions include call instructions and return instructions. Coupled to receive an indication of the return instructions from the return prediction unit, the return prediction unit is configured to predict a return address corresponding to one of the return instructions from the stored return addresses.

The present invention further contemplates a computer system, comprising a microprocessor coupled to a main memory. The microprocessor includes a return prediction structure configured to predict a return address of a return instruction from a plurality of stored return addresses. The stored return addresses comprise return addresses corresponding to a return address of a return instruction from a plurality of stored return addresses. The

	Docum ent ID	U	Title	Current OR
35	US 20030 09754 8 A1	<input checked="" type="checkbox"/>	Context execution in pipelined computer processor	712/228
36	US 20030 09365 5 A1	<input checked="" type="checkbox"/>	Multithread embedded processor with input/output capability	712/228
37	US 20030 08861 0 A1	<input checked="" type="checkbox"/>	Multi-core multi-thread processor	718/107
38	US 20030 06125 8 A1	<input checked="" type="checkbox"/>	Method and apparatus for processing an event occurrence for at least one thread within a multithreaded processor	718/102
39	US 20030 04652 1 A1	<input checked="" type="checkbox"/>	Apparatus and method for switching threads in multi-threading processors	712/228
40	US 20030 04651 7 A1	<input checked="" type="checkbox"/>	Apparatus to facilitate multithreading in a computer processor pipeline	712/214
41	US 20030 03880 8 A1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/506
42	US 20030 03722 8 A1	<input checked="" type="checkbox"/>	System and method for instruction level multithreading scheduling in a embedded processor	712/245
43	US 20030 02383 5 A1	<input checked="" type="checkbox"/>	Method and system to perform a thread switching operation within a multithreaded processor based on dispatch of a quantity of instruction information for a full instruction	712/214
44	US 20030 02383 4 A1	<input checked="" type="checkbox"/>	Method and system to insert a flow marker into an instruction stream to indicate a thread switching operation within a multithreaded processor	712/214
45	US 20030 02365 9 A1	<input checked="" type="checkbox"/>	Method and apparatus for thread switching within a multithreaded processor	718/102
46	US 20030 02365 8 A1	<input checked="" type="checkbox"/>	Method and system to perform a thread switching operation within a multithreaded processor based on detection of the absence of a flow of instruction information for a thread	718/102
47	US 20030 02072 0 A1	<input checked="" type="checkbox"/>	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/506
48	US 20030 01868 7 A1	<input checked="" type="checkbox"/>	Method and system to perform a thread switching operation within a multithreaded processor based on detection of a flow marker within an instruction information	718/102
49	US 20030 01868 6 A1	<input checked="" type="checkbox"/>	Method and system to perform a thread switching operation within a multithreaded processor based on detection of a stall condition	718/102
50	US 20030 01868 5 A1	<input checked="" type="checkbox"/>	Method and system to perform a thread switching operation within a multithreaded processor based on detection of a branch instruction	718/102
51	US 20030 01461 2 A1	<input checked="" type="checkbox"/>	MULTI-THREADED PROCESSOR BY MULTIPLE-BIT FLIP-FLOP GLOBAL SUBSTITUTION	712/215

TABLE 1-continued

Encoding of Start, End and Functional Bits				
Inst.	Start	End	Functional	Meaning
Number	Value	Value	Value	
3-8	0	X	0	byte is prefix
3-8	0	X	1	Mod R/M or SIB byte
3-8	0	X	X	Displacement or immediate data; the second functional bit set in bytes 3-8 indicates
1-8	X	0	X	Not last byte of instruction
1-8	X	1	X	Last byte of instruction

As stated previously, in one embodiment certain instructions within the x86 instruction set may be directly decoded by decode unit 208. These instructions are referred to as "fast path" instructions. The remaining instructions of the x86 instruction set are referred to as "MROM instructions".

MROM instructions are executed by invoking MROM unit 209. More specifically, when an MROM instruction is encountered, MROM unit 209 parses and serializes the instruction into a subset of defined fast path instructions to effectuate a desired operation. A listing of exemplary x86 instructions categorized as fast path instructions as well as a description of the manner of handling both fast path and MROM instructions will be provided further below.

Instruction alignment unit 206 is provided to channel variable byte length instructions from instruction cache 204 to fixed issue positions formed by decode units 208A-208D. In one embodiment, instruction alignment unit 206 independently and in parallel selects instructions from three groups of instruction bytes provided by instruction cache 204 and arranges these bytes into three groups of preliminary issue positions. Each group of issue positions is associated with one of the three groups of instruction bytes. The preliminary issue positions are then merged together to form the final issue positions, each of which is coupled to one of decode units 208.

Before proceeding with a detailed description of the return address prediction mechanism employed with microprocessors 200, general aspects regarding other subsystems employed within the exemplary superscalar microprocessor 200 of FIG. 1 will be described. For the embodiment of FIG. 1, each of the decode units 208 includes decoding circuitry for decoding the predetermined fast path instructions referred to above. In addition, each decode unit 208A-208D routes displacement and immediate data to a corresponding reservation station unit 210A-210D. Output signals from the decode units 208 include bit-encoded execution instructions for the functional units 212 as well as operand address information, immediate data and/or displacement data.

The superscalar microprocessor of FIG. 1 supports out of order execution, and thus employs reorder buffer 216 to keep track of the original program sequence for register read and write operations, to implement register renaming, to allow for speculative instruction execution and branch misprediction recovery, and to facilitate precise exceptions. As will be appreciated by those of skill in the art, a temporary storage

Generally speaking, instruction cache 204 is a high speed cache memory provided to temporarily store instructions prior to their dispatch to decode units 208. In one embodiment, instruction cache 204 is configured to cache up to 32 kilobytes of instruction code organized in lines of 16 bytes each (where each byte consists of 8 bits). During operation, instruction code is provided to instruction cache 204 by prefetching code from a main memory (not shown) through prefetch/prenoted that instruction is noted that instruction cache 204 could be implemented in a set-associative, a fully-associative, or a direct-mapped configuration.

Prefetch/predecode unit 202 is provided to prefetch instruction code from the main memory for storage within instruction cache 204. In one embodiment, prefetch/predecode unit 202 is configured to burst 64-bit wide code from the main memory into instruction cache 204. It is understood that a variety of specific code prefetching techniques and algorithms may be employed by prefetch/predecode unit 202.

As prefetch/predecode unit 202 fetches instructions from the main memory, it generates three predecode bits associated with each byte of instruction code: a start bit, an end bit, and a "functional" bit. The predecode bits form tags indicative of the boundaries of each instruction. The predecode tags may also convey additional information such as whether a given instruction can be decoded directly by decode units 208 or whether the instruction must be executed by invoking a microcode procedure controlled by MROM unit 209, as will be described in greater detail below.

Table 1 indicates one encoding of the predecode tags. As indicated within the table, if a given byte is the first byte of an instruction, the start bit for that byte is set. If the last byte of an instruction, the end bit for that byte is set. If a particular instruction cannot be directly decoded by the decode units 208, the functional bit associated with the first byte of the instruction is set. On the other hand, if the instruction can be directly decoded by the decode units 208, the functional bit associated with the first byte of the instruction is cleared. The functional bit of the first byte of a particular instruction is cleared if the opcode is the first byte, and is set if the opcode is the second byte. It is noted that in situations where the opcode is the second byte, the first byte is a prefix byte. The functional bit values for instruction byte numbers 3-8 indicate whether the byte is a MODRM or an SIB byte, or whether the byte contains displacement or immediate data.

TABLE 1

Encoding of Start, End and Functional Bits				
Inst.	Start	End	Functional	Meaning
Number	Value	Value	Value	
1	1	X	X	Fast decode
1	1	X	0	MROM inst.
2	0	X	0	Opcode is first byte
2	0	X	1	Opcode is this byte, first

	Document ID	U	Title	Current OR
52	US 20030 01422 1 A1	<input checked="" type="checkbox"/>	System and method to avoid resource contention in the presence of exceptions	702/186
53	US 20030 00964 8 A1	<input checked="" type="checkbox"/>	Apparatus for supporting a logically partitioned computer system	711/202
54	US 20030 00526 6 A1	<input checked="" type="checkbox"/>	Multithreaded processor capable of implicit multithreaded execution of a single-thread program	712/220
55	US 20030 00526 3 A1	<input checked="" type="checkbox"/>	Shared resource queue for simultaneous multithreaded processing	712/218
56	US 20030 00526 2 A1	<input checked="" type="checkbox"/>	Mechanism for providing high instruction fetch bandwidth in a multi-threaded processor	712/207
57	US 20020 19917 3 A1	<input checked="" type="checkbox"/>	System, method and article of manufacture for a debugger capable of operating across multiple threads and lock domains	717/129
58	US 20020 18883 2 A1	<input checked="" type="checkbox"/>	Method and apparatus for providing local control of processing elements in a network of multiple context processing elements	712/228
59	US 20020 15699 9 A1	<input checked="" type="checkbox"/>	Mixed-mode hardware multithreading	712/228
60	US 20020 13871 7 A1	<input checked="" type="checkbox"/>	Multiple-thread processor with single-thread interface shared among threads	712/235
61	US 20020 12922 7 A1	<input checked="" type="checkbox"/>	Processor having priority changing function according to threads	712/228
62	US 20020 11660 0 A1	<input checked="" type="checkbox"/>	Method and apparatus for processing events in a multithreaded processor	712/218
63	US 20020 11452 9 A1	<input type="checkbox"/>	Arithmetic coding apparatus and image processing apparatus	382/247
64	US 20020 10399 0 A1	<input type="checkbox"/>	Programmed load precession machine	712/215
65	US 20020 10384 7 A1	<input type="checkbox"/>	Efficient mechanism for inter-thread communication within a multi-threaded computer system	718/107
66	US 20020 09561 4 A1	<input type="checkbox"/>	Method and apparatus for disabling a clock signal within a multithreaded processor	713/500
67	US 20020 09191 5 A1	<input type="checkbox"/>	Load prediction and thread identification in a multithreaded microprocessor	712/225
68	US 20020 08784 4 A1	<input type="checkbox"/>	Apparatus and method for concealing switch latency	712/228

wood cliffs, N.J., 1991, and within the co-pending, commonly assigned patent application entitled "High Performance Superscalar Microprocessor", Ser. No. 08/146,382, filed Oct. 29, 1993 by Witt, et al., abandoned and continued in application Ser. No. 501,243 filed Jul. 10, 1995, now U.S. Pat. No. 5,651,125. These documents are incorporated herein by reference in their entirety.

Reservation station units 210A-210D are provided to temporarily store instruction information to be speculatively executed by the corresponding functional units 212A-212D. As stated previously, each reservation station unit 210A-210D may store instruction information for up to three pending instructions. Each of the four reservation stations 210A-210D contain locations to store bit-encoded execution instructions to be speculatively executed by the corresponding functional unit and the values of operands. If a particular operand is not available, a tag for that operand is provided from reorder buffer 216 and is stored within the corresponding reservation station until the result has been generated (i.e., by completion of the execution of a previous instruction). It is noted that when an instruction is executed by one of the functional units 212A-212D, the result of that instruction is passed directly to any reservation station units 210A-210D that are waiting for that result at the same time the result is passed to update reorder buffer 216 (this technique is commonly referred to as "result forwarding"). Instructions are issued to functional units for execution after the values of any required operand(s) are made available. That is, if an operand associated with a pending instruction within one of the reservation station units 210A-210D has been tagged with a location of a previous result value within reorder buffer 216 which corresponds to an instruction which modifies the required operand, the instruction is not issued to the corresponding functional unit 212 until the operand result for the previous instruction has been obtained. Accordingly, the order in which instructions are executed may not be the same as the order of the original program instruction sequence. Reorder buffer 216 ensures that data coherence is maintained in situations where read-after-write dependencies occur.

In one embodiment, each of the functional units 212 is configured to perform integer arithmetic operations of addition and subtraction, as well as shifts, rotates, logical operations, and branch operations. It is noted that a floating point unit (not shown) may also be employed to accommodate floating point operations.

Each of the functional units 212 also provides information regarding the execution of conditional branch instructions to the branch prediction unit 220. If a branch prediction was incorrect, branch prediction unit 220 flushes instructions subsequent to the mispredicted branch that have entered the instruction processing pipeline, and causes prefetch/predecode unit 200 to fetch the required instructions from such situations, results of instructions in the original program sequence which occur after the mispredicted branch instruction are discarded, including those which were speculatively executed and temporarily stored in load/store unit 222 and reorder buffer 216. Exemplary configurations of suitable branch prediction mechanisms are well known.

Results produced by functional units 212 are sent to the reorder buffer 216 if a register value is being updated, and to the load/store unit 222 if the contents of a memory location is changed. If the result is to be stored in the location, the reorder buffer 216 stores the result in the location reserved for the value of the register when the instruction was decoded. As stated previously, results are also broadcast

location within reorder buffer 216 is reserved upon decode of an instruction that involves the update of a register to thereby store speculative register states. Reorder buffer 216 may be implemented in a first-in-first-out configuration wherein speculative results move to the "bottom" of the buffer as they are validated and written to the register file, thus making room for new entries at the "top" of the buffer. Other specific configurations of reorder buffer 216 are also possible, as will be described further below. If a branch prediction is incorrect, the results of speculatively-executed instructions along the mispredicted path can be invalidated in the buffer before they are written to register file 218.

The bit-encoded execution instructions and immediate data provided at the outputs of decode units 208A-208D are routed directly to respective reservation station units 210A-210D. In one embodiment, each reservation station unit 210A-210D is capable of holding instruction information (i.e., bit encoded execution bits as well as operand values, operand tags and/or immediate data) for up to three pending instructions awaiting issue to the corresponding functional unit. It is noted that for the embodiment of FIG. 1, each decode unit 208A-208D is associated with a dedicated reservation station unit 210A-210D, and that each reservation station unit 210A-210D is similarly associated with a dedicated functional unit 212A-212D. Accordingly, four dedicated "issue positions" are formed by decode units 208, reservation station units 210 and functional units 212. Instructions aligned and dispatched to issue position 0 through decode unit 208A are passed to reservation station unit 210A and subsequently to functional unit 212A for execution. Similarly, instructions aligned and dispatched to decode unit 208B are passed to reservation station unit 210B and into functional unit 212B, and so on.

Upon decode of a particular instruction, if a required operand is a register location, register address information is routed to reorder buffer 216 and register file 218 simultaneously. Those of skill in the art will appreciate that the x86 register file includes eight 32 bit real registers (i.e., typically referred to as EAX, EBX, ECX, EDX, EBP, ESI, EDI and ESP). Reorder buffer 216 contains temporary storage locations for results which change the contents of these registers to thereby allow out of order execution. A temporary storage location of reorder buffer 216 is reserved for each instruction which, upon decode, is determined to modify the contents of one of the real registers. Therefore, at various points during execution of a particular program, reorder buffer 216 may have one or more locations which contain the speculatively executed contents of a given register. If following decode of a given instruction it is determined that reorder buffer 216 has a previous location or locations assigned to a register used as an operand in the given instruction, the reorder buffer 216 forwards to the corresponding reservation station either: 1) the value in the most recently assigned location, or a tag for the most recently assigned location if the value has not yet been produced by the functional unit that will eventually execute the previous instruction. If the reorder buffer has a location reserved for a given register, the operand value (or tag) is provided from reorder buffer 216 rather than from register file 218. If there is no location reserved for a required register in reorder buffer 216, the value is taken directly from register file 218. If the operand corresponds to a memory location, the operand value is provided to the reservation station unit through load/store unit 222.

Details regarding suitable reorder buffer implementations may be found within the publication "Superscalar Microprocessor Design" by Mike Johnson, Prentice-Hall, Engle-

	Docum ent ID	U	Title	Current OR
69	US 20020 08784 3 A1	<input type="checkbox"/>	Method and apparatus for reducing components necessary for instruction pointer generation in a simultaneous multithreaded processor	712/228
70	US 20020 08784 0 A1	<input type="checkbox"/>	Method for converting pipeline stalls to pipeline flushes in a multithreaded processor	712/219
71	US 20020 08783 5 A1	<input type="checkbox"/>	Method and apparatus for improving dispersal performance in a processor through the use of no-op ports	712/215
72	US 20020 08337 3 A1	<input type="checkbox"/>	Journaling for parallel hardware threads in multithreaded processor	714/38
73	US 20020 07812 2 A1	<input type="checkbox"/>	Switching method in a multi-threaded processor	718/102
74	US 20020 05603 7 A1	<input type="checkbox"/>	Method and apparatus for providing large register address space while maximizing cycletime performance for a multi-threaded register file set	712/215
75	US 20020 05459 4 A1	<input type="checkbox"/>	Non-blocking, multi-context pipelined processor	370/389
76	US 20020 04632 5 A1	<input type="checkbox"/>	Buffer memory management in a system having multiple execution entities	711/122
77	US 20020 03841 6 A1	<input type="checkbox"/>	System and method for reading and writing a thread state in a multithreaded central processing unit	712/228
78	US 20020 01386 1 A1	<input type="checkbox"/>	Method and apparatus for low overhead multithreaded communication in a parallel processing environment	719/313
79	US 20020 00266 7 A1	<input type="checkbox"/>	System and method for instruction level multithreading in an embedded processor using zero-time context switching	712/228
80	US 20010 05645 6 A1	<input type="checkbox"/>	PRIORITY BASED SIMULTANEOUS MULTI-THREADING	718/103
81	US 20010 04977 0 A1	<input type="checkbox"/>	BUFFER MEMORY MANAGEMENT IN A SYSTEM HAVING MULTIPLE EXECUTION ENTITIES	711/129
82	US 20010 04746 8 A1	<input type="checkbox"/>	Branch and return on blocked load or store	712/228
83	US 20010 03744 5 A1	<input type="checkbox"/>	Cycle count replication in a simultaneous and redundantly threaded processor	712/216
84	US 20010 02951 5 A1	<input type="checkbox"/>	Method and apparatus for configuring arbitrary sized data paths comprising multiple context processing elements	708/232
85	US 20010 00475 5 A1	<input type="checkbox"/>	MECHANISM FOR FREEING REGISTERS ON PROCESSORS THAT PERFORM DYNAMIC OUT-OF-ORDER EXECUTION OF INSTRUCTIONS USING RENAMING REGISTERS	712/217

IV

	Docum ent ID	U	Title	Current OR
86	US 67218 73 B2	<input type="checkbox"/>	Method and apparatus for improving dispersal performance in a processor through the use of no-op ports	712/215
87	US 66979 35 B1	<input type="checkbox"/>	Method and apparatus for selecting thread switch events in a multithreaded processor	712/228
88	US 66944 25 B1	<input type="checkbox"/>	Selective flush of shared and other pipeline stages in a multithread processor	712/216
89	US 66943 47 B2	<input type="checkbox"/>	Switching method in a multi-threaded processor	718/108
90	US 66752 85 B1	<input type="checkbox"/>	Geometric engine including a computational module without memory contention	712/201
91	US 66751 92 B2	<input type="checkbox"/>	Temporary halting of thread execution until monitoring of armed events to memory location identified in working registers	718/107
92	US 66718 27 B2	<input type="checkbox"/>	Journaling for parallel hardware threads in multithreaded processor	714/38
93	US 66683 17 B1	<input type="checkbox"/>	Microengine for parallel processor architecture	712/245
94	US 66586 55 B1	<input type="checkbox"/>	Method of executing an interpreter program	717/139
95	US 66585 51 B1	<input type="checkbox"/>	Method and apparatus for identifying splittable packets in a multithreaded VLIW processor	712/24
96	US 66584 47 B2	<input type="checkbox"/>	Priority based simultaneous multi-threading	718/103
97	US 66503 30 B2	<input type="checkbox"/>	Graphics system and method for processing multiple independent execution threads	345/506
98	US 66402 99 B1	<input type="checkbox"/>	Method and apparatus for arbitrating access to a computational engine for use in a video graphics controller	712/245
99	US 66309 35 B1	<input type="checkbox"/>	Geometric engine including a computational module for use in a video graphics controller	345/522
100	US 66292 36 B1	<input type="checkbox"/>	Master-slave latch circuit for multithreaded processing	712/228
101	US 66256 54 B1	<input type="checkbox"/>	Thread signaling in multi-threaded network processor	709/230
102	US 66248 18 B1	<input type="checkbox"/>	Method and apparatus for shared microcode in a multi-thread computation engine	345/522
103	US 66112 76 B1	<input type="checkbox"/>	Graphical user interface that displays operation of processor threads over time	345/772
104	US 66067 04 B1	<input type="checkbox"/>	Parallel multithreaded processor with plural microengines executing multiple threads each microengine having loadable microcode	712/248
105	US 65947 55 B1	<input type="checkbox"/>	System and method for interleaved execution of multiple independent threads	712/239
106	US 65913 57 B2	<input type="checkbox"/>	Method and apparatus for configuring arbitrary sized data paths comprising multiple context processing elements	712/18
107	US 65879 06 B2	<input type="checkbox"/>	Parallel multi-threaded processing	710/240
108	US 65781 37 B2	<input type="checkbox"/>	Branch and return on blocked load or store	712/228

	Docum ent ID	U	Title	Current OR
109	US 65739 00 B1	<input type="checkbox"/>	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/537
110	US 65678 39 B1	<input type="checkbox"/>	Thread switch control in a multithreaded processor system	718/103
111	US 65670 84 B1	<input type="checkbox"/>	Lighting effect computation circuit and method therefore	345/426
112	US 65534 79 B2	<input type="checkbox"/>	Local control of multiple context processing elements with major contexts and minor contexts	712/16
113	US 65429 91 B1	<input type="checkbox"/>	Multiple-thread processor with single-thread interface shared among threads	712/228
114	US 65429 21 B1	<input type="checkbox"/>	Method and apparatus for controlling the processing priority between multiple threads in a multithreaded processor	718/108
115	US 65359 05 B1	<input type="checkbox"/>	Method and apparatus for thread switching within a multithreaded processor	718/108
116	US 65325 09 B1	<input type="checkbox"/>	Arbitrating command requests in a parallel multi-threaded processing system	710/240
117	US 65264 98 B1	<input type="checkbox"/>	Method and apparatus for retiming in a network of multiple context processing elements	712/11
118	US 65078 62 B1	<input type="checkbox"/>	Switching method in a multi-threaded processor	718/107
119	US 64969 25 B1	<input type="checkbox"/>	Method and apparatus for processing an event occurrence within a multithreaded processor	712/244
120	US 64937 41 B1	<input type="checkbox"/>	Method and apparatus to quiesce a portion of a simultaneous multithreaded central processing unit	718/107
121	US 64704 43 B1	<input type="checkbox"/>	Pipelined multi-thread processor selecting thread instruction in inter-stage buffer based on count information	712/205
122	US 64704 22 B2	<input type="checkbox"/>	Buffer memory management in a system having multiple execution entities	711/129
123	US 64571 16 B1	<input type="checkbox"/>	Method and apparatus for controlling contexts of multiple context processing elements in a network of multiple context processing elements	712/16
124	US 64386 71 B1	<input type="checkbox"/>	Generating partition corresponding real address in partitioned mode supporting system	711/173
125	US 63780 65 B1	<input type="checkbox"/>	Apparatus with context switching capability	712/228
126	US 63742 86 B1	<input type="checkbox"/>	Real time processor capable of concurrently running multiple independent JAVA machines	718/108
127	US 63634 75 B1	<input type="checkbox"/>	Apparatus and method for program level parallelism in a VLIW processor	712/206
128	US 63570 16 B1	<input type="checkbox"/>	Method and apparatus for disabling a clock signal within a multithreaded processor	713/601
129	US 63518 08 B1	<input type="checkbox"/>	Vertically and horizontally threaded processor with multidimensional storage for storing thread data	712/228
130	US 63493 63 B1	<input type="checkbox"/>	Multi-section cache with different attributes for each section	711/129
131	US 63413 47 B1	<input type="checkbox"/>	Thread switch logic in a multiple-thread processor	712/228

one embodiment, branch prediction unit 220 conveys the misprediction signal. In still another embodiment, the functional unit 212 which detects the misprediction signal. Upon receipt of the branch misprediction signal, comparator block 260 compares the call and return tags stored within return stack storage 252 to the branch tag conveyed upon branch tag bus 280. If a particular call or return tag is found to be subsequent to the branch tag in program order, then an associated invalidate signal is asserted upon invalidate bus 282 to return stack control unit 254. Invalidate bus 282 includes an invalidate signal for each call tag and return tag within return stack storage 252, and the signal may be asserted according to a matching comparison between the associated call or return tag and a branch tag from branch tag bus 280 or 276. It is noted that call tags and return tags stored within return stack storage 252 are conveyed to comparator block 260 upon call tag bus 284 and return tag bus 286, respectively.

Upon receipt of an asserted invalidate signal, the associated CV or RV bit within return stack storage 252 is reset. In this manner, call and return instructions subsequent to the mispredicted branch are removed from return stack storage 252. Once a call instruction is retired by reorder buffer 216, the associated call tag is invalid. Therefore, if an invalidate signal is asserted for a call instruction for which the CRV bit is set, then the CV bit is left unmodified. Additionally, storage entries for which the call instruction have been invalidated no longer store valid information. Similar to removing entries for which the return instruction has been retired, the storage locations are shifted and the pointer value adjusted to delete the invalid entries from return stack storage 252.

It is noted that call and return instructions for which the target address is mispredicted are treated as mispredicted branch instructions for purposes of this discussion. It is further noted that a mispredicted address for a return instruction may be indicative of a fake return instruction. Although conditions other than the existence of a fake return instruction may be the cause of the misprediction, it is difficult to distinguish the various conditions within return prediction unit 250. Therefore, the entire return stack is invalidated when a mispredicted return instruction is detected. Additionally, if the contents of instruction cache 204 are invalidated, then the return stack is invalidated. Instruction cache 204 may be invalidated due to a task switch, for example. The contents of return stack storage 252 may be invalidated for the new task, and therefore the return stack is invalidated.

Return stack storage 252 includes a finite number of entries, and may therefore become full before any entries are deleted. When a call instruction is detected and return stack storage 252 is full of valid entries, then the entry stored at the "bottom" of the stack (i.e., the entry allocated prior to other entries within the stack) is deleted. In one embodiment, the pointer upon pointer bus 254 wraps around to the bottom storage location within return stack storage 252 and allocates that location to the newly detected call instruction. In another embodiment, all storage locations within return stack storage 252 are shifted down one location (similar to when an entry is deleted due to return instruction retirement) and the top storage location is allocated to the new return instruction. The pointer upon pointer bus 264 is unmodified for this embodiment.

It is noted that certain instructions within various micro-processor architectures may generate an "exception". An exception causes program execution to jump to an exception handling routine, similar to an interrupt. If an instruction generates an exception, the branch tag conveyed with the return stack is recovered according to the mispredicted branch recovery sequence noted above. It is further noted that, when multiple call and/or return instructions are detected simultaneously by decode units 208, return stack control unit 254 is configured to select the first instruction detected in program order. The other call and/or return instructions will be purged from the instruction processing pipeline due to the first instructions being detected. It is still further noted that the above discussion describes signals as being "asserted". A signal may be defined as being asserted when it conveys a value indicative of a particular piece of information. A particular signal may be defined to be asserted when it conveys a binary one value or, alternatively, when it conveys a binary zero value. A second embodiment of return prediction unit 250 is contemplated in which call and return instructions are detected in decode units 208 but not in branch prediction unit 220. For this embodiment, functionality is similar to the above description with the exception of detection of call and return instructions in branch prediction unit 220. Additionally, an embodiment is contemplated in which call, return and branch tags may be tags from reorder buffer 216 indicative of the position within reorder buffer 216 storing the associated instruction.

Turning now to FIG. 2A, another embodiment of return stack storage 252 (return stack storage 252A) is shown. The return PC, call tag, and return tag are included, as well as the CV, RV, and CRV bits similar to return stack storage 252 shown in FIG. 2. Additionally, return stack storage 252A includes an IXC bit and a START bit. The IXC bit is set if the call tag is associated with an INT instruction, and is cleared if the call tag is associated with a CALL instruction. When a prediction is made by return prediction unit 250 for an RET instruction, the storage location nearest the top of return stack storage 252A in which the RV bit is clear and the IXC bit is set is used as the prediction. Similarly, when a prediction is made by return prediction unit 250 for a RET instruction, the storage location nearest the top of return stack storage 252A in which both the RV and IXC bits are clear is used as the prediction. In this manner, CALL-RET pairs may be separated from INT-RET pairs.

It is noted that the RET instruction is not only used in conjunction with the interrupt instruction. Additionally, the RET instruction is used to return from asynchronous interrupts. For example, microprocessor 200 may include an interrupt pin which may be asserted by external hardware to interrupt microprocessor 200. An interrupt service routine is executed by microprocessor 200 in response to the interrupt, often ending in an RET instruction to cause microprocessor 200 to return to the interrupted instruction sequence. Because of this alternative usage of the RET instruction, RET has a higher probability of being mispredicted than the RET instruction. It may be performance limiting to invalidate the entire return stack upon a mispredicted RET instruction. The START bit is set when microprocessor 200 enters an interrupt service routine. If the interrupt service routine is entered due to an asynchronous interrupt, then an entry is allocated with the program count value of the instruction interrupted as the return PC and the START bit is set. If the interrupt service routine is entered due to an INT instruction, the START bit is set when the INT instruction allocates an entry. If an RET instruction is mispredicted, the entries within return stack storage 252A between the top entry and the entry nearest the top for which the START bit

	Docum ent ID	U	Title	Current OR
132	US 63306 61 B1	<input type="checkbox"/>	Reducing inherited logical to physical register mapping information between tasks in multithread system using register group identifier	712/228
133	US 63145 11 B1	<input type="checkbox"/>	Mechanism for freeing registers on processors that perform dynamic out-of-order execution of instructions using renaming registers	712/217
134	US 62984 31 B1	<input type="checkbox"/>	Banked shadowed register file	712/28
135	US 62956 00 B1	<input type="checkbox"/>	Thread switch on blocked load or store using instruction thread field	712/228
136	US 62601 50 B1	<input type="checkbox"/>	Foreground and background context controller setting processor to power saving mode when all contexts are inactive	713/323
137	US 62567 75 B1	<input type="checkbox"/>	Facilities for detailed software performance analysis in a multithreaded processor	717/127
138	US 62533 13 B1	<input type="checkbox"/>	Parallel processor system for processing natural concurrencies and method therefor	712/226
139	US 62437 36 B1	<input type="checkbox"/>	Context controller having status-based background functional task resource allocation capability and processor employing the same	718/108
140	US 62267 35 B1	<input type="checkbox"/>	Method and apparatus for configuring arbitrary sized data paths comprising multiple context processing elements	712/18
141	US 62232 74 B1	<input type="checkbox"/>	Power-and speed-efficient data storage/transfer architecture models and design methodologies for programmable or reusable multi-media processors	712/34
142	US 62232 08 B1	<input type="checkbox"/>	Moving data in and out of processor units using idle register/storage functional units	718/108
143	US 62162 20 B1	<input type="checkbox"/>	Multithreaded data processing method with long latency subinstructions	712/219
144	US 62125 44 B1	<input type="checkbox"/>	Altering thread priorities in a multithreaded processor	718/103
145	US 62125 42 B1	<input type="checkbox"/>	Method and system for executing a program within a multiscalar processor by processing linked thread descriptors	718/102
146	US 62054 68 B1	<input type="checkbox"/>	System for multitasking management employing context controller having event vector selection by priority encoding of contex events	718/108
147	US 61700 51 B1	<input type="checkbox"/>	Apparatus and method for program level parallelism in a VLIW processor	712/225
148	US 61611 66 A	<input type="checkbox"/>	Instruction cache for multithreaded processor	711/125
149	US 61346 53 A	<input type="checkbox"/>	RISC processor architecture with high performance context switching in which one context can be loaded by a co-processor while another context is being accessed by an arithmetic logic unit	712/228
150	US 61345 78 A	<input type="checkbox"/>	Data processing device and method of operation with context switching	718/100
151	US 61227 19 A	<input type="checkbox"/>	Method and apparatus for retiming in a network of multiple context processing elements	712/15
152	US 61087 60 A	<input type="checkbox"/>	Method and apparatus for position independent reconfiguration in a network of multiple context processing elements	711/203
153	US 61051 27 A	<input type="checkbox"/>	Multithreaded processor for processing multiple instruction streams independently of each other by flexibly controlling throughput in each instruction stream	712/215

illustrate the dynamics of the present return stack structure in more detail. Beginning at arrow 440, an instruction stream including instructions INS0 through INS1 are executed. Instructions INS0 through INS9, similar to FIG. 3, represent instructions which are not branch, Call, or Ret instructions. Following INS1 is a branch instruction Jump, with a branch tag of one. In this example, decimal numbers are used for branch tags. However, many other numbering schemes may be used for branch tags. The Jump instruction is predicted taken, and so instruction execution begins at the target of the Jump instruction (arrow 402).

At the target of the Jump instruction is an instruction INS2 followed by a Ret instruction. The Ret instruction is assigned the next available branch tag, which in the example numbering scheme shown is a branch tag value of two. Instruction execution then begins at the predicted target of the Ret instruction (arrow 404). The remainder of the instruction stream is similar, following consecutively through arrows 406, 408, 410, 414, 416, 418, 422, 426, and 428. Branch, Call, and Ret instructions are assigned consecutive branch tags as shown in FIG. 4A. Each instruction is numbered for reference in FIGS. 4B through 4E below. It is noted that Ret instruction 409 returns to the instruction subsequent to Call instruction 405. This relationship is shown by dotted line 412. Similarly, Ret instruction 415 returns to the instruction subsequent to Call instruction 413 (line 420) and Ret instruction 417 returns to the instruction subsequent to Call instruction 411 (line 424).

Turning now to FIG. 4B, the state of return stack storage 252 prior to the execution of INS0 is shown (i.e., the state of the return stack storage at arrow 400 shown in FIG. 4A). Three valid entries are shown (reference numbers 440, 442, and 444), with return addresses A, B, and C (respectively). In this example, letters are used as exemplary return addresses for brevity. A return address is in fact a multi-bit number. In one embodiment, a return address comprises 32 bits. A call tag of zero is stored in each entry, although the entries storing return addresses A and B have their respective CRV bits set, and so the respective call tags are no longer valid for those instructions. Each entry has its CV bit set, indicating that the entries are valid. None of the RV bits are set, and so entries 440, 442, and 444 have not yet been used as return address predictions. Each entry includes a dash in the return tag field to indicate that the return tags are not valid. The pointer upon pointer bus 362 is shown pointing to the third entry of return stack storage 252. Third entry 444 is the current top of the valid entries within return stack storage 252.

Turning now to FIG. 4C, the contents of return stack storage 252 are shown at the time of fetching Jump instruction 407 (i.e., at arrow 406 in FIG. 4A). A fourth entry has been added from the state shown in FIG. 4B (return address D, call tag of three, reference number 446). Additionally, entry 444 includes a valid return tag value of two, and the RV bit is set. The contents of return stack storage 252 may be explained by considering that the first call or return encountered is Ret instruction 403. At the time the Ret instruction is fetched, the top entry in return stack storage 252 is entry 444. Since the RV bit of entry 444 is clear, return address C is predicted for Ret instruction 403. The RV bit is then set, and the return tag of Ret instruction 403 (i.e., the value two) is stored into the return tag field of entry 444. The next Call instruction 405 is encountered at the fetch of Call instruction 405. Fourth entry 446 is allocated at the fetch of Call instruction 405, and the call tag is set to the call tag of Call instruction 405 (i.e., the value of three). The pointer is now shown pointing to fourth entry 446.

Turning now to FIG. 4A, a second exemplary instruction sequence is shown. This instruction sequence is used to In both cases, the correct return address is predicted.

ing Call A is used as the return address prediction for Ret A. The address of the instruction immediately following Call A is used as the return address prediction for Ret A. B has yet to be retired), that entry has been used as a return address prediction. Therefore, the second from the top entry B is still the top entry of the return stack storage (since Ret entry which has not been used as a prediction. Although Call return address prediction is again made according to the top instruction retiring, the Ret A instruction is fetched. The During a subsequent clock cycle, prior to the Ret B address following Call B.

Therefore, the return address for Ret B is predicted to be the since no return address predictions have yet been made. above). In this example, the entry chosen is the top entry, address prediction is used as the prediction address (as noted stack storage 252 which has not been used as a return When Ret B is fetched, the entry nearest the top of return pointer bus 264), and the Call A entry is second from the top. the return stack storage 252 (as defined by the pointer upon prediction unit 250 as well. The Call B entry is at the top of fetched and the return address is placed within return prediction unit 250. During a later clock cycle, Call B is fetched and the return address is placed within return and Ret A instructions. During a clock cycle, Call A is may be gained by predicting the return address of the Ret B executed by a supercalar microprocessor, then performance If the exemplary instruction sequence shown in FIG. 3 is call and return instructions are associated.

stack storage 252 is well suited to the LIFO manner in which be paired with a return instruction, etc. it is noted that return manner. The last call instruction to be executed is the first to paired with call instructions in a last-in, first-out (LIFO) executed from that point forward. Return instructions are the instruction following the Call A instruction in memory is recently executed call instruction (Call A, arrow 308). Therefore, Ret A is defined to return to the second most second return instruction encountered in the example. A is immediately following Call B in this example, and is the locations storing the Call B instruction. The instruction Ret stored in memory locations contiguous to the memory B, arrow 306). In other words, Ret B fetches an instruction following the most recently executed call instruction (Call defined to return to the instruction in memory immediately Ret B is the first return instruction encountered, and so is retired.

INS6 are executed, and a return instruction Ret B is encountered third set of contiguous instructions labeled INS5 through causing instruction execution to transfer to yet another address specified by the Call B instruction (arrow 304). A executed. A second call instruction Call B is encountered, continuous instructions labeled INS3 through INS4 are executed. A set of address specified by Call A (arrow 302), and a set of A is encountered. Instruction execution transfers to an through INS2 are executed, and then a call instruction Call instruction stream of contiguous instructions labeled INS0 branch, Call, or Ret instructions. Beginning at arrow 300, an INS0 through INS8 represent instructions which are not instructions are used. It is noted that instructions labeled discussion, for brevity, it is assumed that no fake return tion unit 250 in predicting addresses. In the following is shown to further highlight the operation of return prediction now to FIG. 3, an exemplary instruction sequence interrupt service routine may be invalidated.

return stack storage 252A which is associated with the is set are invalidated. In this manner, only the portion of

	Docum ent ID	U	Title	Current OR
154	US 61050 51 A	<input type="checkbox"/>	Apparatus and method to guarantee forward progress in execution of threads in a multithreaded processor	718/103
155	US 61015 99 A	<input type="checkbox"/>	System for context switching between processing elements in a pipeline of processing elements	712/228
156	US 60921 75 A	<input type="checkbox"/>	Shared register storage mechanisms for multithreaded computer systems with out-of-order execution	712/23
157	US 60887 88 A	<input type="checkbox"/>	Background completion of instruction and associated fetch request in a multithread processor	712/205
158	US 60790 08 A	<input type="checkbox"/>	Multiple thread multiple data predictive coded parallel processing system and method	712/11
159	US 60761 57 A	<input type="checkbox"/>	Method and apparatus to force a thread switch in a multithreaded processor	712/228
160	US 60731 59 A	<input type="checkbox"/>	Thread properties attribute vector based thread selection in multithreading processor	718/103
161	US 60617 10 A	<input type="checkbox"/>	Multithreaded processor incorporating a thread latch register for interrupt service new pending threads	718/107
162	US 60527 08 A	<input type="checkbox"/>	Performance monitoring of thread switch events in a multithreaded processor	718/108
163	US 60187 59 A	<input type="checkbox"/>	Thread switch tuning tool for optimal performance in a computer processor	718/108
164	US 59499 94 A	<input type="checkbox"/>	Dedicated context-cycling computer with timed context	712/228
165	US 59336 27 A	<input type="checkbox"/>	Thread switch on blocked load or store using instruction thread field	712/228
166	US 59241 20 A	<input type="checkbox"/>	Method and apparatus for maximizing utilization of an internal processor bus in the context of external transactions running at speeds fractionally greater than internal transaction times	711/141
167	US 59151 23 A	<input type="checkbox"/>	Method and apparatus for controlling configuration memory contexts of processing elements in a network of multiple context processing elements	712/16
168	US 59139 25 A	<input type="checkbox"/>	Method and system for constructing a program including out-of-order threads and processor and method for executing threads out-of-order	712/206
169	US 59077 02 A	<input type="checkbox"/>	Method and apparatus for decreasing thread switch latency in a multithread processor	718/108
170	US 58988 64 A	<input type="checkbox"/>	Method and system for executing a context-altering instruction without performing a context-synchronization operation within high-performance processors	712/228
171	US 58871 66 A	<input type="checkbox"/>	Method and system for constructing a program including a navigation instruction	718/102
172	US 58812 77 A	<input type="checkbox"/>	Pipelined microprocessor with branch misprediction cache circuits, systems and methods	712/239
173	US 58729 85 A	<input type="checkbox"/>	Switching multi-context processor and method overcoming pipeline vacancies	710/1
174	US 58257 70 A	<input type="checkbox"/>	Multiple algorithm processing on a plurality of digital signal streams via context switching	370/378
175	US 57421 80 A	<input type="checkbox"/>	Dynamically programmable gate array with multiple contexts	326/40

decade units is further coupled to instruction alignment unit 508, and a set 512 of reservation station/functional units is coupled to a load/store unit 514 and to a reorder buffer 516. A register file unit 518 is finally shown coupled to reorder buffer 516, and a data cache 522 is shown coupled to load/store unit 514.

Processor 500 limits the addressing mechanism used in the x86 to achieve both regular simple form of addressing as well as high clock frequency execution. It also targets 32-bit O/S and applications. Specifically, 32-bit flat addressing is employed where all the segment registers are mapped to all 4GB of physical memory. The starting address being 0000-0000 hex and their limit address being FFFF hex. The setting of this condition will be detected within processor 500 as one of the conditions will allow the collection of accelerated data paths and instructions to be enabled. The absence of this condition on instruction issue and a trapping to MROM condition will cause a serialization of 32-bit flat addressing will be enabled. The absence of this condition on instruction issue and a trapping to MROM space.

Another method to ensure that a relatively high clock frequency may be accommodated is to limit the number of memory address calculation schemes to those that are simple to decode and can be decoded within a few bytes. We are also interested in supporting addressing that fits into our other goals, i.e., regular instruction decoding.

As a result, the x86 instruction types that are supported for load/store operations are:

push	[implied ESP - 4]
pop	[implied ESP + 4]
call	[implied ESP + 8]
ret	[implied ESP - 8]
load	[base + 8-bit displacement]
store	[base + 8-bit displacement]
oper.	[EBP + 8-bit displacement]
oper.	[EAX + 8-bit displacement]

The block diagram of FIG. 6 shows the pipeline for calculating addressing within processor 500. It is noted that base + 9/32 bit displacement takes 1 cycle, where using an index register takes 1 more cycle of delay in calculating the address. More complicated addressing than these requires invoking an MROM routine to execute.

An exemplary listing of the instruction sub-set supported by processor 500 as fast path instructions is provided below. All other x86 instructions will be executed as micro-ROM sequences of fast path instructions or extensions to fast path instructions.

The standard x86 instruction set is very limited in the number of registers it provides. Most RISC processors have 32 or greater general purpose registers, and many important variables can be held during and across procedures or processes during normal execution of routines. Because there are so few registers in the x86 architecture and most are not general purpose, a large percentage of operations are moves to and from memory. RISC architectures also incorporate 3 operand addressing to prevent moves from occurring of register values that are desired to be saved instead of overwritten.

The x86 instruction set uses a set of registers that can trace its history back to the 8080. Consequently there are few registers, many side effects, and sub-registers within registers. This is because when moving to 16-bit, or 32-bit operators, mode bits were added and the lengths of the registers were extended instead of expanding the size of the register file. Modern compiler technology can make use of

Turning next to FIG. 4D, the state of return stack storage 252 is shown after fetching Call instruction 413 (i.e. at arrow 416 shown in FIG. 4A). Two additional entries (reference numbers 448 and 450) have been added with respect to the state shown in FIG. 4C, and entry 446 has been updated. Entry 446 is updated at the fetching of Ret instruction 409, which includes a return tag value of five. Entry 446 is the top of return stack storage 252 at the time Ret instruction 409 is fetched, and the RV bit of entry 446 is clear. Therefore, return address D is used as the return address prediction and the RV bit of entry 446 is set. Call instructions 411 and 413, respectively, cause the allocation of entries 448 and 450. Therefore, entry 448 includes a call tag value of six and entry 450 includes a call tag value of seven.

Turning now to FIG. 4E, the state of return stack storage 252 at the end of the exemplary instruction stream shown in FIG. 4A (i.e. at arrow 418 as shown in FIG. 4A). Yet another entry 452 has been added, and entries 448, 450, and 442 have been updated. Entry 450 is updated at the fetch of Ret instruction 415, because entry 450 is at the top of stack storage 252 at the time Ret instruction 415 is fetched. As shown in FIG. 4E, return address F is used as a prediction for Ret instruction 415 and the return tag associated with Ret instruction 415 is stored into entry 450. Similarly, Ret instruction 417 causes the update of entry 448. Entry 448 is not at the top of return stack storage 252, but the entry which is at the top (entry 450) has already been used as a return address prediction for Ret instruction 415. Therefore, entry 448 is used as the return address prediction for Ret instruction 417.

Additionally, Ret instruction 419 is fetched and a return address prediction is formulated. At the time Ret instruction 419 is fetched, entries 450, 448, 446, and 444 have previously been used as return address predictions. Therefore, entry 442 is the entry of return stack storage 252 nearest the top of the stack which has not yet been used as a return address prediction. Return address B is prediction, and entry 442 is updated with the return tag associated with Ret instruction 419 (i.e. a return tag value of ten). Call instruction 421 is then decoded and causes entry 452 to be added to the top of return stack storage 252.

The exemplary code sequence shown in FIG. 4A may be used to illustrate the recovery of return prediction unit 250 from mispredicted branches. As an example, consider Jump instruction 407 being mispredicted and the misprediction being detected at arrow 416, such that FIG. 4D shows the state of return stack storage 252. Jump instruction 407 includes a branch tag of four, and therefore any instructions having call and return tags subsequent to a branch tag value of four within return stack storage 252 are invalidated. It is noted that call tags which are associated with retired call instructions are not invalidated. As shown in FIG. 4D, the return tag of entry 446 and the call tags of entries 450 and 448 are invalidated. As a second example, consider Jump instruction 407 not being determined to be mispredicted until return stack storage 252 achieves the state shown in FIG. 4E. In this example, the return tags of entries 450, 448, 446, and 442 and the call tags of entries 452, 450, and 448 are invalidated. It is noted that call instructions whose target addresses are incorrectly predicted operate similarly with respect to return stack storage 252.

Turning next to FIGS. 5-6, details regarding various aspects of another embodiment of a superscalar microprocessor are next considered. FIG. 5 is a block diagram of a processor 500 including an instruction cache 502 coupled to a prefetch/predecode unit 504, to a branch prediction unit 506, and to an instruction alignment unit 508. A set 510 of

	Docum ent ID	U	Title	Current OR
176	US 56525 81 A	<input type="checkbox"/>	Distributed coding and prediction by use of contexts	341/51
177	US 55749 39 A	<input type="checkbox"/>	Multiprocessor coupling system with integrated compile and run time scheduling for parallelism	712/24
178	US 55509 93 A	<input type="checkbox"/>	Data processor with sets of two registers where both registers receive identical information and when context changes in one register the other register remains unchanged	712/229
179	US 55505 40 A	<input type="checkbox"/>	Distributed coding and prediction by use of contexts	341/51
180	US 54044 69 A	<input type="checkbox"/>	Multi-threaded microprocessor architecture utilizing static interleaving	712/215
181	US 53613 37 A	<input type="checkbox"/>	Method and apparatus for rapidly switching processes in a computer system	712/228
182	US 53576 17 A	<input type="checkbox"/>	Method and apparatus for substantially concurrent multiple instruction thread processing by a single pipeline processor	712/245
183	US 53496 87 A	<input type="checkbox"/>	Speech recognition system having first and second registers enabling both to concurrently receive identical information in one context and disabling one to retain the information in a next context	704/231
184	US 53197 92 A	<input type="checkbox"/>	Modem having first and second registers enabling both to concurrently receive identical information in one context and disabling one to retain the information in a next context	712/228
185	US 53197 89 A	<input type="checkbox"/>	Electromechanical apparatus having first and second registers enabling both to concurrently receive identical information in one context and disabling one to retain the information in a next context	712/228
186	US 53136 48 A	<input type="checkbox"/>	Signal processing apparatus having first and second registers enabling both to concurrently receive identical information in one context and disabling one to retain the information in a next context	712/228
187	US 51797 34 A	<input type="checkbox"/>	Threaded interpretive data processor	712/1
188	US 51426 77 A	<input type="checkbox"/>	Context switching devices, systems and methods	718/108
189	US 48477 55 A	<input type="checkbox"/>	Parallel processing method and apparatus for increasing processing throughout by parallel processing low level instructions having natural concurrencies	712/203

-continued-

```

!jump unconditional
16-bit operations
prefetch/move reg/reg
prefetch/move reg/mem
prefetch/arithmetic operations reg/reg, reg/mem
prefetch/logical operations reg/reg, reg/mem
prefetch/push
prefetch/pop

```

When executing 32-bit code under flat addressing, these instructions almost always fall within 1-8 bytes in length, which is in the same rough range of the aligned, accelerated fast path instructions.

Accelerated instructions are defined as fast-path instructions between 1 and 8 bytes in length. It noted that it is possible that the start/end positions predicated reflect multiple x86 instructions, for instance 2 or 3 pushes that are predicated in a row may be treated as one accelerated instruction that consumes 3 bytes.

When a cache line is fetched from the instruction cache, it moves into an instruction alignment unit which looks for start bytes within narrow ranges. The instruction alignment unit uses the positions of the start bytes of the instructions to dispatch the instructions to four issue positions. Instructions are dispatched such that each issue position accepts the first valid start byte within its range along with subsequent bytes.

A multiplexer in each decoder looks for the end byte associated with each start byte, where an end byte can be no more than seven bytes away from a start byte. The mechanism to scan for a constant value in an instruction over four bytes in length may be given an extra pipeline stage due to the amount of time potentially required.

Note that instructions included in the subset of accelerated instructions, and which are over four bytes in length, always have a constant as the last 1/2/4 bytes. This constant is usually not needed until the instruction is issued to a functional unit, and therefore the determination of the constant value can be delayed in the pipeline. The exception is an instruction requiring an eight-bit displacement for an address calculation. The eight-bit displacement for stack-relative operations is always the third byte after the start byte, so this field will always be located within the same decoder as the rest of the instruction.

It is possible that a given cache line can have more instructions to issue than can be accommodated by the four entry positions contained in each line of the four issue reorder buffer. If this occurs, the four issue reorder buffer allocates a second line in the buffer as the remaining instructions are dispatched. Typically, in 32-bit application and O/S code, the average instruction length is about three bytes. The opcode is almost always the first two bytes, with the third byte being a sib byte specifying a memory address (if included), and the fourth byte being a 16-bit data prefix.

The assumption in the processor 500 alignment hardware is that if the average instruction length is three, then four dedicated issue positions and decoders assigned limited byte ranges should accommodate most instructions found within 16-byte instruction cache lines. If very dense decoding occurs (i.e., lots of one and two byte instructions), several lines are allocated in the four issue reorder buffer for the results of instructions contained in a few lines of the instruction cache. The fact that these more compact instructions are still issued in parallel and at a high clock frequency more than compensates for having some decoder positions potentially idle.

large register sets and have a much smaller percentage of loads and stores. The effect of these same compilers is to compile to the x86. The actual x86 registers are often relegated to temporary registers for a few clock cycles while the real operation destinations are in memory.

FIG. 7 shows a programmer's view of the x86 register file. One notes from this organization that there are only 8 registers, and few are general purpose. The first four registers, EAX, EDI, ECX, and EBX, have operand sizes of 8, 16, or 32-bits depending on the mode of the processor or instruction. The final 4 registers were added with the 8086 and extended with the 386. Because there are so few real registers, they tend to act as holding positions for the passing of variables to and from memory.

The important thing to note is that when executing x86 instructions, one must be able to efficiently handle 8, 16, and 32-bit operands. If one is trying to execute multiple x86 instructions in parallel, it is not enough to simply multi-port the register file. This is because there are too few registers and all important program variables must be held in memory on the stack or in a fixed location.

RISC designs employ regular instruction decoding along natural boundaries to achieve very high clock frequencies and also with a small number of pipeline stages even for very wide issue processors. This is possible because finding a large number of instructions and their opcodes is relatively straightforward, since they are always at fixed boundaries. As stated previously, this is much more difficult in an x86 processor where there are variable byte instruction formats, as well as prefix bytes and SIB bytes that can effect the length and addressing/data types of the original opcode.

Processor 500 employs hardware to detect and send simple instructions to fixed issue positions, where the range of bytes that a particular issue position can use is limited. This may be compensated for by adding many issue positions that each instruction cache line can assume in parallel. Once the instructions are aligned to a particular issue position, the net amount of hardware required to decode common instructions is not significantly greater than that of a RISC processor, allowing equivalent clock frequencies to be achieved. Processor 500 achieves high frequency, wide issue, and limited pipeline depth by limiting the instructions executed at high frequency to a sub-set of the x86 instructions under the conditions of 32-bit flat addressing.

The results of executing instructions are returned to the corresponding entry in the reorder buffer. If a store, the store is held in speculative state in front of the data cache in a store buffer, from which point it can be speculatively forwarded from. The reorder buffer then can either cancel this store or allow it to writeback to the data cache when the line is released.

The following set of instructions probably comprise 90% of the dynamically executed code for 32-bit applications:

```

8/32-bit operations
move reg/reg reg/mem
arithmetic operations reg/mem reg/reg logical
operations reg/reg reg/mem push
push
pop
call/return
load effective address
jump cc

```

	Document ID	U	Title	Current OR
1	US 20040 07390 5 A1	<input type="checkbox"/>	Method and apparatus to quiesce a portion of a simultaneous multithreaded central processing unit	718/101
2	US 20040 07377 8 A1	<input type="checkbox"/>	Parallel processor architecture	712/220
3	US 20040 05488 0 A1	<input type="checkbox"/>	Microengine for parallel processor architecture	712/245
4	US 20040 03475 9 A1	<input type="checkbox"/>	Multi-threaded pipeline with context issue rules	712/1
5	US 20030 14515 9 A1	<input type="checkbox"/>	SRAM controller for parallel processor architecture	711/104
6	US 20030 10594 4 A1	<input type="checkbox"/>	Method and apparatus to quiesce a portion of a simultaneous multithreaded central processing unit	712/220
7	US 20030 09754 8 A1	<input type="checkbox"/>	Context execution in pipelined computer processor	712/228
8	US 20030 03722 8 A1	<input type="checkbox"/>	System and method for instruction level multithreading scheduling in a embedded processor	712/245
9	US 20030 00526 2 A1	<input type="checkbox"/>	Mechanism for providing high instruction fetch bandwidth in a multi-threaded processor	712/207
10	US 20020 12922 7 A1	<input type="checkbox"/>	Processor having priority changing function according to threads	712/228
11	US 20020 08337 3 A1	<input type="checkbox"/>	Journaling for parallel hardware threads in multithreaded processor	714/38
12	US 20020 05603 7 A1	<input type="checkbox"/>	Method and apparatus for providing large register address space while maximizing cycletime performance for a multi-threaded register file set	712/215
13	US 20020 03841 6 A1	<input type="checkbox"/>	System and method for reading and writing a thread state in a multithreaded central processing unit	712/228
14	US 20020 02170 7 A1	<input type="checkbox"/>	Method and apparatus for non-speculative pre-fetch operation in data packet processing	370/412
15	US 20020 01848 6 A1	<input type="checkbox"/>	Context selection and activation mechanism for activating one of a group of inactive contexts in a processor core for servicing interrupts	370/463
16	US 20020 00266 7 A1	<input type="checkbox"/>	System and method for instruction level multithreading in an embedded processor using zero-time context switching	712/228
17	US 20010 05205 3 A1	<input type="checkbox"/>	Stream processing unit for a multi-streaming processor	711/138

Overview of the Processor 500 Instruction Cache (ICache)
This section describes the instruction cache organization, the Pro-
cessor 500 instruction cache has basic features including the
ICSTORE, ICTAGV, ICNXTBLK, ICNNTI, ICALIGN, and ICPC, and ICPCRED. Highlights are: the pre-decode bits per
byte of instructions are 3 bits; the branch prediction
increases to 2 targets, 2 different types of branch prediction
techniques (bimodal and global) are implemented; the X86
and the pre-decode logic eliminates many serialization con-
ditions. Processor 500 executes the X86 instructions directly
with a few instructions requiring two Rops, the BYTEQ is
configured for fast scanning of instructions, and instructions
are aligned to 4 decode units. The pre-decode data is
separate in a block called ICPCDAT, instead of inside the
ICSTORE. The pre-fetch buffers are added to the ICSTORE
to write instructions directly into the array, and the prefixes
are not modified. All branches are detected during pre-
decoding. Unconditional branches are taken during pre-
decoding and aligning of instructions to the decode units. A
return stack is implemented for CALL/RETURN instruc-
tions. Way prediction is implemented to read the current
block and fetch the next block because the tag comparison
and branch prediction do not resolve until the second cycle.
The scanning for 4 instructions is done from ICPCDAT before
instructions must be partially decoded for the 2-rop during
prioritizing and aligning of instructions to decode units. The
early decoding including decoding for unconditional
branches, operand addresses, flags, displacement and imme-
diate fields of the instruction. The CMASTER takes care of
the replacement algorithm for the ICache and sends the way
associative along with the data to the pre-fetch buffer. This
section includes signal lists, timings and implementation
issues for the ICache and all sub-blocks.
The ICache size is 32K bytes with 8-way set associative.
The ICache is linearly addressed. The number of pipeline
stages is 9. ICache will have more than one clock cycle to
read and align the instructions to the decode units. The row
decoding of index address is calculated in first half of ICLK.
The data, tag, pre-decode, and predicting information are
read in by the end of ICLK. In the next cycle, the data are
selected by the TAGHTS and latched. The pre-decode data
are scanned to generate the controls to the multiplexers for
aligning and sending the instructions to the decode units and
ROM units. A part of the scanning logic is done in parallel
with the tag comparison. The scanning and alignment of
instructions takes two clock cycles. The decode units can
start decoding in the second half of the third clock. The
ICache includes a way-prediction which can be done in a
single clock using the ICNXTBLK target. The branch pre-
diction includes bimodal and global branch prediction which
takes two clock cycles. The timing from fetching, scanning,
aligning, decoding, and muxing of instructions to decode
units are shown in FIG. 8.
Throughout this documentation, a discussion of the layout
organization is included in each section. The array is orga-
nized into many sets, and each set has its own decoder. The
decoder is in the center of the set.
Signal list
SRBB(31:0)—I/O from SRB indicates the special register
address for the array or data transferring to/from the SRB.
SRB_VAL—Input from SRB indicates a special register
instruction is on the SRBB.
ICTAR_VAL—Output to SRB indicates completion of the
special register instruction, for read the data is on the
SRBB.

IRESET—Global signal used to reset ICACHE block.
Clears all state machines to Idle/Reset.
IDECAVIC—Global signal from FIROB. Used to indicate
that an interrupt or trap is being taken. Effect on ICache is
to clear all pre-fetch or access in progress, and set all state
machines to Idle/Reset.
EXCEPTION—Global input from FIROB indicates that an
interrupt or trap is being taken including
re-synchronization. Effect on ICache and FUs is to clear
all instructions in progress.
REGTRAP—Global input from FIROB, one cycle after
EXCEPTION, indicates that the trap is initiated with new
entry point or new PC is driven.
INVBHREG—Input from FIROB to invalidate the branch
holding register. The branch mis-prediction is speculative,
an early branch can be mis-predicted at a later time.
CS32X16—Input from LSSEC indicates operand and
address size from the D bit of the segment descriptor of
the code segment register. If set, 32-bit; if clear, 16-bit.
SUPERV—Input from LSSEC indicates the supervisor
mode or user mode of the current accessed instruction.
TR12DIC—Input from SRB indicates that all un-cached
instructions must be fetched from the external memory.
SRBINVALV—Input from SRB to invalidate the ICache by
clear all valid bits.
INSTRDY—Input from BIU to indicates the valid external
fetched instruction is on the INSB(63:0) bus.
INSTFLT—Input from BIU to indicates the valid but faulted
external fetched instruction is on the INSB(63:0) bus.
INSB(63:0)—Input from external buses for fetched instruc-
tion to the ICache.
L2_IC_ALIAS—Input from CMASTER indicates the
instruction is in the ICache with different mapping. The
CMaster provides the way associative and new super-
visor bit. The LV will be set in this case.
PREFETCHCOL(2:0)—Input from CMASTER indicates the
way associative for writing of the ICTAGV.
UPDFPC—Input from FIROB indicate that a new Fetch PC
has been detected. This signal accompanies the FPC for
the ICache to begin access the cache arrays.
FPC(31:0)—Input from FIROB as the new PC for branch
correction path.
BPC(11:0)—Input from FIROB indicates the PC index and
byte-pointer of the branch instruction which has been
mis-predicted for updating the ICNXTBLK. This index
must be compared to the array index for exact recovery of
the global shift register
BRNMISP—Input from the Branch execution of the FU
indicates a branch mis-prediction. The ICache changes its
state machine to access a new PC and clears all pending
instructions.
BRNTAKEN—Input from FIROB indicate the status of the
mis-prediction. This signal must be gated with UPDFPC.
BRNTAG(3:0)—Input from FIROB indicates the instruc-
tion byte for updating the branch prediction in the
ICNXTBLK.
FPC_TY—Input for FIROB indicates the type of address
that is being passed to the ICache.
HIDISP(1:0)—Output to ICache indicates all instructions
of the first (bit 0) and/or the second (bit 1) 8-byte of the
current line has been dispatched to decode units.
REFRESH2—Input from ICache indicates current line of
instructions will be refreshed and not accept new instruc-
tions from ICache.
MROMEND—Input from MENG indicates completion of
the MROM.
DOUSEFI(4:0)

	Docum ent ID	U	Title	Current OR
18	US 66944 25 B1	<input type="checkbox"/>	Selective flush of shared and other pipeline stages in a multithread processor	712/216
19	US 66751 92 B2	<input type="checkbox"/>	Temporary halting of thread execution until monitoring of armed events to memory location identified in working registers	718/107
20	US 66718 27 B2	<input type="checkbox"/>	Journaling for parallel hardware threads in multithreaded processor	714/38
21	US 66683 17 B1	<input type="checkbox"/>	Microengine for parallel processor architecture	712/245
22	US 66338 65 B1	<input type="checkbox"/>	Multithreaded address resolution system	707/3
23	US 66067 04 B1	<input type="checkbox"/>	Parallel multithreaded processor with plural microengines executing multiple threads each microengine having loadable microcode	712/248
24	US 64937 41 B1	<input type="checkbox"/>	Method and apparatus to quiesce a portion of a simultaneous multithreaded central processing unit	718/107
25	US 64704 43 B1	<input type="checkbox"/>	Pipelined multi-thread processor selecting thread instruction in inter-stage buffer based on count information	712/205
26	US 64703 76 B1	<input type="checkbox"/>	Processor capable of efficiently executing many asynchronous event tasks	718/108
27	US 64271 96 B1	<input type="checkbox"/>	SRAM controller for parallel processor architecture including address and command queue and arbiter	711/158
28	US 63634 75 B1	<input type="checkbox"/>	Apparatus and method for program level parallelism in a VLIW processor	712/206
29	US 62956 00 B1	<input type="checkbox"/>	Thread switch on blocked load or store using instruction thread field	712/228
30	US 61700 51 B1	<input type="checkbox"/>	Apparatus and method for program level parallelism in a VLIW processor	712/225
31	US 61190 75 A	<input type="checkbox"/>	Method for estimating statistics of properties of interactions processed by a processor pipeline	702/186
32	US 60947 15 A	<input type="checkbox"/>	SIMD/MIMD processing synchronization	712/20
33	US 60731 59 A	<input type="checkbox"/>	Thread properties attribute vector based thread selection in multithreading processor	718/103
34	US 59665 28 A	<input type="checkbox"/>	SIMD/MIMD array processor with vector processing	712/222
35	US 59637 46 A	<input type="checkbox"/>	Fully distributed processing memory element	712/20
36	US 59637 45 A	<input type="checkbox"/>	APAP I/O programmable router	712/13
37	US 59499 94 A	<input type="checkbox"/>	Dedicated context-cycling computer with timed context	712/228
38	US 58812 77 A	<input type="checkbox"/>	Pipelined microprocessor with branch misprediction cache circuits, systems and methods	712/239
39	US 58782 41 A	<input type="checkbox"/>	Partitioning of processing elements in a SIMD/MIMD array processor	712/203
40	US 58706 19 A	<input type="checkbox"/>	Array processor with asynchronous availability of a next SIMD instruction	712/20

DOWRFL(4:0)—Output to FIROB indicates the type of flag uses/writes for this instruction of decode unit 0:
 xx1 CF-carry flag,
 lxx SF-sign, ZF-zero, PF-parity, and AF-auxiliary carry
D1WFL(4:0)—Output to FIROB indicates the type of flag uses/writes for this instruction of decode unit 1:
D2USEFL(4:0)
D2WFL(4:0)—Output to FIROB indicates the type of flag uses/writes for this instruction of decode unit 2:
D3USEFL(4:0)
D3WFL(4:0)—Output to FIROB indicates the type of flag uses/writes for this instruction of decode unit 3:
RDOPTR1(5:0)—Indicates the register address for operand 1 of decode unit 0. The MROM is responsible to send bit 5:3 for the MROM register.
RD1PTR1(5:0)—Indicates the register address for operand 1 of decode unit 1. The MROM is responsible to send bit 5:3 for the MROM register.
RD2PTR1(5:0)—Indicates the register address for operand 1 of decode unit 2. The MROM is responsible to send bit 5:3 for the MROM register.
RD3PTR1(5:0)—Indicates the register address for operand 1 of decode unit 3. The MROM is responsible to send bit 5:3 for the MROM register.
RD0PTR2(5:0)—Indicates register address for operand 2 of decode unit 0. The MROM is responsible to send bit 5:3 for the MROM register.
RD1PTR2(5:0)—Indicates register address for operand 2 of decode unit 1. The MROM is responsible to send bit 5:3 for the MROM register.
RD2PTR2(5:0)—Indicates register address for operand 2 of decode unit 2. The MROM is responsible to send bit 5:3 for the MROM register.
RD3PTR2(5:0)—Indicates register address for operand 2 of decode unit 3. The MROM is responsible to send bit 5:3 for the MROM register.
IDXDAT1(0)—Output to indicates the data size information. 01-byte, 10-half word, 11-word, 00-not used.
ICBTAG2(3:0)—Output to Idecode indicates the position of the first target branch instruction with respect to the global shift register in case of branch mis-prediction. The branch can be taken or non-taken, branch tag must be sent with all branch instruction.
ICBTAG3(3:0)—Output to Idecode indicates the position of the second target branch instruction with respect to the global shift register in case of branch mis-prediction. The branch can be taken or non-taken, branch tag must be sent with all branch instruction.
UNJMP(3:0)—Output indicates the unconditional branch instruction needs to calculate target address.
BRNTKN(3:0)—Output indicates which decode unit has a predicted taken branch. The operand steering uses this signal to latch and send BTADDR(31:0) to the functional unit.
BRNNST(3:0)—Output indicates which decode unit has a global branch prediction. The operand steering uses this signal to latch and send ICBTAG1(3:0) and ICBTAG2(3:0) to the functional units.
CALDEC(3:0)—Output to FIROB indicates the CALL instruction is detected, the return stack should be updated with the PC address of instruction after CALL. The information is latched for mis-predicted CALL branch.
RETDEC(3:0)—Output to FIROB indicates a RETURN instruction is detected. The information is latched for mis-predicted RETURN branch.

ICPREF(9:0)—Output to Idecode and MROM indicates the encoded prefix byte. The two most significant bits are repeat prefixes for MROM.
IC2ROP(3:0)—Output to decode unit 0 indicates 2-top instruction. Bit 3 indicates the first rop or second rop of the 2-rop instruction, bit 2 indicates POP instruction, bit 1 indicates the MUL instruction, and bit 0 indicates the SIB-byte instruction.
NODESTR(3:0)—Output to FIROB indicates no destination for the first rop of the SIB-byte instruction.
DEPVG(3:1)—Output to FIROB indicates forced dependency tag on the first instruction; the second rop of the SIB-byte instruction.
REFRESH2—Input from Idecode indicates current line of instructions will be refreshed and not accept new instructions from lcache.
IBI(19:0)—Output indicates the combined instruction line for dispatching to decode units.
MROMEN—Input from MENG indicates the micro-instructions is sent to Idecode instead of the lcache.
MOUSEFL(4:0)
M0WRF1(4:0)—Input from MENG indicates the type of flag used/written for this micro-instruction of decode unit 0:
 xx1 CF-carry flag,
 lxx OF-overflow flag,
 lxx SF-sign, ZF-zero, PF-parity, and AF-auxiliary carry
M1USEFL(4:0)
M1WRF1(4:0)—Input from MENG indicates the type of flag used/written for this micro-instruction of decode unit 1:
M2USEFL(4:0)
M2WRF1(4:0)—Input from MENG indicates the type of flag used/written for this micro-instruction of decode unit 2:
M3USEFL(4:0)
M3WRF1(4:0)—Input from MENG indicates the type of flag used/written for this micro-instruction of decode unit 3.
MINS(63:0)—Input from MENG indicates the displacement and immediate field of micro-instruction being sent to decode 0.
MINS1(63:0)—Input from MENG indicates the displacement and immediate field of micro-instruction being sent to decode 1.
MINS2(63:0)—Input from MENG indicates the displacement and immediate field of micro-instruction being sent to decode 2.
MINS3(63:0)—Input from MENG indicates the displacement and immediate field of micro-instruction being sent to decode 3.
MR0OPC(7:0)—Input from MENG to decode unit 0 indicates the opcode byte.
MR1OPC(7:0)—Input from MENG to decode unit 1 indicates the opcode byte.
MR2OPC(7:0)—Input from MENG to decode unit 2 indicates the opcode byte.
MR3OPC(7:0)—Input from MENG to decode unit 3 indicates the opcode byte.
MR0EOP(2:0)—Input from MENG to decode unit 0 indicates the extended opcode field.
MR1EOP(2:0)—Input from MENG to decode unit 1 indicates the extended opcode field.
MR2EOP(2:0)—Input from MENG to decode unit 2 indicates the extended opcode field.
MR3EOP(2:0)—Input from MENG to decode unit 3 indicates the extended opcode field.

	Docum ent ID	U	Title	Current OR
41	US 58483 73 A	<input type="checkbox"/>	Computer aided map location system	701/200
42	US 58420 31 A	<input type="checkbox"/>	Advanced parallel array processor (APAP)	712/23
43	US 58094 50 A	<input type="checkbox"/>	Method for estimating statistics of properties of instructions processed by a processor pipeline	702/186
44	US 57940 59 A	<input type="checkbox"/>	N-dimensional modified hypercube	712/10
45	US 57650 11 A	<input type="checkbox"/>	Parallel processing system having a synchronous SIMD processing with processing elements emulating SIMD operation using individual instruction streams	712/20
46	US 57615 23 A	<input type="checkbox"/>	Parallel processing system having asynchronous SIMD processing and data parallel coding	712/20
47	US 57548 71 A	<input type="checkbox"/>	Parallel processing system having asynchronous SIMD processing	712/20
48	US 57520 67 A	<input type="checkbox"/>	Fully scalable parallel processing system having asynchronous SIMD processing	712/16
49	US 57349 21 A	<input type="checkbox"/>	Advanced parallel array processor computer package	712/10
50	US 57179 44 A	<input type="checkbox"/>	Autonomous SIMD/MIMD processor memory elements	712/20
51	US 57179 43 A	<input type="checkbox"/>	Advanced parallel array processor (APAP)	712/20
52	US 57130 37 A	<input type="checkbox"/>	Slide bus communication functions for SIMD/MIMD array processor	702/33
53	US 57109 35 A	<input type="checkbox"/>	Advanced parallel array processor (APAP)	712/20
54	US 57088 36 A	<input type="checkbox"/>	SIMD/MIMD inter-processor communication	712/20
55	US 56258 36 A	<input type="checkbox"/>	SIMD/MIMD processing memory element (PME)	709/214
56	US 55903 45 A	<input type="checkbox"/>	Advanced parallel array processor (APAP)	712/11
57	US 55881 52 A	<input type="checkbox"/>	Advanced parallel processor including advanced support hardware	712/16
58	US 43251 20 A	<input type="checkbox"/>	Data processing system	711/202

MROSS(1:0)—Input from **MENG** to decode unit 0 indicates the scale factor of the SIB byte.

MRISS(1:0)—Input from **MENG** to decode unit 1 indicates the scale factor of the SIB byte.

MR2SS(1:0)—Input from **MENG** to decode unit 2 indicates the scale factor of the SIB byte.

MR3SS(1:0)—Input from **MENG** to decode unit 3 indicates the scale factor of the SIB byte.

ICMROM—Output to **MENG** indicates the current instruction is **MROM**. The **MROM** instruction may take two cycles to read the **IB**, **ICEND**, and **ICFUNC**.

ICPCTTAR—Output to **Idcode** indicates is **ICPC1** a branch target of a previous instruction which is a predicted taken branch instruction.

ICPC2TAR—Output to **Idcode** indicates is **ICPC2** a branch target of a previous instruction which is a predicted taken branch instruction.

ICPC1(3:1:0)—Output to **Idcode** indicates the current line **PC** of the first instruction in the 4 issued instructions to pass along with the instruction to **FIROB**.

ICPC2(3:1:0)—Output to **Idcode** indicates the current line **PC** of a second instruction which crosses the 16-byte boundary or branch target in the 4 issued instructions to pass along with the instruction to **FIROB**.

ICPOS(4:0)—Output to decode unit 0 indicates the **PC**'s byte position of the next instruction. Bit 4 indicates the next instruction is on the next line.

ICPOS1(4:0)—Output to decode unit 1 indicates the **PC**'s byte position of the next instruction. Bit 4 indicates the next instruction is on the next line.

ICPOS2(4:0)—Output to decode unit 2 indicates the **PC**'s byte position of the next instruction. Bit 4 indicates the next instruction is on the next line.

ICPOS3(4:0)—Output to decode unit 3 indicates the **PC**'s byte position of the next instruction. Bit 4 indicates the next instruction is on the next line.

BTAG1N(3:0)—Output indicates the position of the first target branch instruction for a new line with respect to the global shift register in case of branch mis-prediction.

BTAG2N(3:0)—Output indicates the position of the second target branch instruction for a new line with respect to the global shift register in case of branch mis-prediction.

BTAKEN1(1:0)—Output to decode units and **ICFPC** indicates a predicted taken branch instruction from **PTAKEN**, **BVAL1**. Bit 0 is the last line and bit 1 is new line.

BTAKEN2(1:0)—Output to decode units and **ICFPC** indicates a predicted taken branch instruction from **PTAKEN**, **BVAL2**. Bit 0 is the last line and bit 1 is new line.

ICERROR—Output, indicates an exception has occurred on an instruction pre-fetched, the type of exception (**TLB**-miss, page-fault, illegal opcode, external bus error) will also be asserted.

INSPFET—Output to **BIU** and **CMASSTER** requests instruction fetching from the previous incremented address, the pre-fetch buffer in the **Icache** has space for a new line from external memory.

ICAD(3:1:0)—Output to **MMU** indicates a new fetch **PC** request to external memory.

MROSS(1:0)—Input from **MENG** to decode unit 0 indicates the scale factor of the SIB byte.

MRISS(1:0)—Input from **MENG** to decode unit 1 indicates the scale factor of the SIB byte.

MR2SS(1:0)—Input from **MENG** to decode unit 2 indicates the scale factor of the SIB byte.

MR3SS(1:0)—Input from **MENG** to decode unit 3 indicates the scale factor of the SIB byte.

ICMROM—Output to **MENG** indicates the current instruction is **MROM**. The **MROM** instruction may take two cycles to read the **IB**, **ICEND**, and **ICFUNC**.

ENDINST—Input from **ICPRE** indicates that pre-decoding is completed for the current instruction. The byte position of the branch instruction is from **STARTPTR**. The selected instruction from **IB** should be sent to decode unit 0.

ICVAL(3:0)—Output to **Idcode** indicates valid instructions. **NOOP** is generated for invalid instruction.

ICOPC(7:0)—Output to decode unit 0 indicates the opcode byte.

IC1OPC(7:0)—Output to decode unit 1 indicates the opcode byte.

IC2OPC(7:0)—Output to decode unit 2 indicates the opcode byte.

IC3OPC(7:0)—Output to decode unit 3 indicates the opcode byte.

IC0EOP(2:0)—Output to decode unit 0 indicates the extended opcode field.

IC1EOP(2:0)—Output to decode unit 1 indicates the extended opcode field.

IC2EOP(2:0)—Output to decode unit 2 indicates the extended opcode field.

IC3EOP(2:0)—Output to decode unit 3 indicates the extended opcode field.

IC0SS(1:0)—Output to decode unit 0 indicates the scale factor of the SIB byte.

IC1SS(1:0)—Output to decode unit 1 indicates the scale factor of the SIB byte.

IC2SS(1:0)—Output to decode unit 2 indicates the scale factor of the SIB byte.

IC3SS(1:0)—Output to decode unit 3 indicates the scale factor of the SIB byte.

DISPTR1(6:0)—Output to decode unit 1 indicates the displacement pointer and size. Bits 2:0 is the pointer to the 8-byte block, bit 6:5 is the size, and bit 4:3 indicates which 8-byte block. Bit 6:5=00 indicates no displacement.

DISPTR2(6:0)—Output to decode unit 2 indicates the displacement pointer and size. Bits 2:0 is the pointer to the 8-byte block, bit 6:5 is the size, and bit 4:3 indicates which 8-byte block. Bit 6:5=00 indicates no displacement.

DISPTR3(6:0)—Output to decode unit 3 indicates the displacement pointer and size. Bits 2:0 is the pointer to the 8-byte block, bit 6:5 is the size, and bit 4:3 indicates which 8-byte block. Bit 6:5=00 indicates no displacement.

MMMPTR1(4:0)—Output to decode unit 1 indicates the displacement pointer and size. Bits 2:0 is the pointer to the 8-byte block, and bit 4:3 indicates which 8-byte block. Decoding of all opcodes is needed to detect immediate field.

MMMPTR2(4:0)—Output to decode unit 2 indicates the displacement pointer and size. Bits 2:0 is the pointer to the 8-byte block, and bit 4:3 indicates which 8-byte block. Decoding of all opcodes is needed to detect immediate field.

MMMPTR3(4:0)—Output to decode unit 3 indicates the displacement pointer and size. Bits 2:0 is the pointer to the 8-byte block, and bit 4:3 indicates which 8-byte block. Decoding of all opcodes is needed to detect immediate field.